НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О. В. Коваль
(підпис) (ініціали та прізвище)

«____» _____ 2020 р.

# ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 121 “ Інженерія програмного забезпечення “

на тему: «Текстова новела з вільним вибором та з елементами штучного інтелекту»

Виконала: студентка 4 курсу, групи ТІ-61

Халімон Олександра Олександрівна _____
(прізвище, ім’я, по-батькові) (підпис)

Керівник к.т.н., доцент Ходаковський О. В. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2020

ЗАВДАННЯ

**Національний технічний університет України**

**“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет: теплоенергетичний
Кафедра: автоматизації проектування енергетичних процесів і систем
Рівень вищої освіти: перший ,бакалаврський
Напрям підготовки: 121 “Інженерія програмного забезпечення“

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
«_____» _____ 2020 р.

ЗАВДАННЯ
на дипломну роботу студенту
Халімон Олександрі Олександрівні
(прізвище, ім’я, по батькові)

1. Тема роботи: Текстова новела з вільним вибором та з елементами штучного інтелекту

Керівник роботи: Ходаковський Олексій Володимирович, к.т.н., доцент
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”_ __ _”___ __ 2020 р. № __

2. Строк подання студентом роботи:

3. Вихідні дані до роботи: мова програмування Android Java та JavaScript. Середовище розробки Android Studio та Firebase.

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити): провести аналітичний огляд аналогічних програмних засобів; проаналізувати обрані підходи та технології, обґрунтувати вибір інструментів для різного застосування в дипломній роботі; розробити архітектуру платформи, розробити схеми та діаграми, які пояснюють структуру проекту; описати процеси функціонування платформи; розробити прототип мобільного додатку; розробити мобільний додаток та платформу для зберігання даних; провести тестування розробленого продукту; сформулювати висновки по проведеній роботі.

5. Перелік ілюстративного матеріалу: Актуальність роботи; Мета роботи; Аналіз аналогів; Виявлені вимоги до програмного засобу; методи і засоби розробки програмного продукту; Аналіз результатів розробки програмного продукту; Демонстрація; Висновки.

6. Дата видачі завдання ”25” листопада 2020 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів виконання дипломної роботи | Термін виконання етапів роботи | Примітки |
|---|---|---|---|
| 1. | Вивчення та аналіз завдання | 15.02.2020-20.02.2020 | |
| 2 | Вивчення літератури | 21.02.2020-25.02.2020 | |
| 3. | Аналітичний огляд існуючих аналогічних засобів | 01.03.2020-03.03.2020 | |
| 4. | Вибір інструментів та технологій | 05.03.2020-15.03.2020 | |
| 5. | Розробка архітектури проекту | 16.03.2020-11.04.2020 | |
| 6. | Розробка процесів проекту | 12.04.2020-22.04.2020 | |
| 7. | Розробка інтерфейсу проекту | 23.04.2020-04.05.2020 | |
| 8. | Розробка мобільного додатку | 05.05.2020-17.05.2020 | |
| 9. | Розробка пояснювальної записки | 18.05.2020-08.06.2020 | |
| 10. | Передзахист | 09.06.2020 | |
| 11. | Захист | 16.06.2020 | |

Студент _____ Халімон О. О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Ходаковський О. В.
(підпис) (прізвище та ініціали)

# АНОТАЦІЯ

Обсяг даної дипломної роботи складає 85 сторінок, 17 рисунків, 6 таблиць, 5 додатків, 26 бібліографічних найменувань за «Переліком посилань».

Метою даного дипломного проекту було проектування та розробка мобільного медичного сервісу для видаленої взаємодії користувача із державними медичними закладами.

Було досліджено продукти-аналоги та виділено основні недоліки. Було оглянуто процес прототипування програмного продукту та виділено переваги, завдяки яким прототип продукту є невід'ємною частиною процесу розробки програмного забезпечення. Також було розглянуто концепцію noSQL сховищ та хмарних технологій. Було обрано сховище Firebase Realtime Database. Було розглянуто UML-діаграми та їх роль в розробці програмного забезпечення. Вихідний продукт має клієнт-серверну архітектуру. Клієнтську частину освітньої платформи реалізовано у середовищі Android Studio мовою Java із частковим використанням концепції Material Design.

У результаті роботи здійснена програмна реалізація медичного сервісу для видаленої взаємодії користувача із державними медичними закладами.

Ключові слова: мобільний додаток, хмарні технології, база даних, інтерфейс, досвід користувача, векторна графіка, мобільний пристрій, нереляційна база даних, магазин, діаграма, архітектура програмного забезпечення.

# ABSTRACT

The volume of this document is 85 pages, 17 drawings, 6 tables, 5 appendices, 26 bibliographic names according to the “List of references”.

The purpose of this diploma project was to design and develop a mobile medical service for remote user interaction with government medical institutions.

Analogous products were investigated and major disadvantages identified. The process of prototyping a software product was reviewed and the benefits of making a product prototype an integral part of the software development process were highlighted. The concept of noSQL repositories and cloud technologies was also considered. The Firebase Realtime Database repository was selected. UML diagrams and their role in software development were discussed. The original product has a client-server architecture. The client side of the educational platform was implemented in Android Studio in Java with partial use of the Material Design concept.

As a result, the software implementation of medical service for remote user interaction with state medical institutions was carried out.

Keywords: mobile application, cloud technologies, database, interface, user experience, vector graphics, mobile device, non-relational database, shop, diagram, software architecture.

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

*UX* – емоції та ставлення людини щодо використання певного товару, системи чи послуги. Він включає практичні, досвідні, афективні, змістовні та цінні аспекти взаємодії людини та комп'ютера та володіння продуктами. Крім того, воно включає сприйняття людиною системних аспектів, таких як корисність, простота використання та ефективність.

*JSON* – відкритий стандартний формат файлу та формат обміну даними, який використовує зручний для розуміння людиною текст для зберігання та передачі об'єктів даних, що складаються з пар атрибутів-значень та типів даних масиву (або будь-яке інше значення, яке може бути типізованим).

*Material Design* – стиль графічного дизайну інтерфейсів програмного забезпечення і додатків, розроблений компанією *Google.*

*UML* – мова графічного опису для об'єктного моделювання в області розробки програмного забезпечення, для моделювання бізнес-процесів, системного проектування та відображення організаційних структур.

*Draw.io* – сервіс для графічного відображення діаграм, моделей графіків тощо.

*Illustrator* – додаток для спеціалізованого використання, який надає інструменти для роботи з векторною графікою.

*UI* – інтерфейс, що забезпечує передачу інформації між користувачем-людиною і програмно-апаратними компонентами комп'ютерної системи

*GUI* – система засобів для взаємодії користувача з комп'ютером, заснована на представленні всіх доступних користувачеві системних об'єктів і функцій у вигляді графічних компонентів екрану (вікон, значків, меню, кнопок, списків і т. п.).

*IDE* – комплекс програмних засобів, який використовується програмістами для розробки програмного забезпечення.

*Android* – операційна система для смартфонів, планшетів, електронних книг, цифрових програвачів, наручних годинників, фітнес-браслетів, ігрових приставок, ноутбуків, нетбуків, смартбуків тощо.

*Android Studio* – це інтегроване середовище розробки (*IDE*) для роботи з платформою *Android*.

*NoSQL* – термін, що позначає ряд підходів, спрямованих на реалізацію систем управління базами даних, що мають суттєві відмінності від моделей, що використовуються в традиційних реляційних базах даних.

*key-value* – парадигма зберігання даних, призначена для зберігання, вилучення та управління асоціативними масивами, структура даних, більш відома сьогодні як словник чи хеш-таблиця.

*Firebase Realtime Database* – база даних, що дозволяє створювати бази даних на основі концепції «ключ-значення», дозволяючи безпечний доступ до бази даних безпосередньо з застосунку клієнта.

*HTTP-запит* – протокол прикладного рівня передачі даних спочатку – у вигляді гіпертекстових документів в форматі «*HTML*», зараз використовується для передачі довільних даних.

*Firebase Function* – безсерверний фреймворк, який дозволяє автоматично запускати резервний код у відповідь на події, викликані функціями *Firebase* та *HTTPS*-запитами.

*Firebase Crashlytics* – модуль аналізу помилок у режимі реального часу, який допомагає відстежувати, визначати пріоритети та виправляти проблеми зі стабільністю, які погіршують якість програми.

*Firebase Test Lab* – хмарна інфраструктура тестування додатків.

*Play Market – Google Play*, раніше *Android Market*, американська послуга цифрового розповсюдження, керована та розроблена *Google*. Він служить офіційним магазином додатків для операційної системи *Android*, дозволяючи користувачам переглядати та завантажувати додатки, розроблені разом із набором програмного забезпечення для *Android* та опублікованими через *Google*.

# ЗМІСТ

# ВСТУП

На сьогоднішній час розвиток технологій дозволяє встановити з’єднання з будь-яким закладом державної установи, таким чином, звільняючи людину від черг, паперової роботи та зберігаючи час. Наразі у світі створено велику кількість універсальних платформ та сервісів, які дозволяють суспільству обмінюватися знаннями, вміннями, досвідом тощо.

У більшості випадків такі платформи не забезпечені функціоналом, який не є більшою мірою затребуваним сучасним суспільством або не принесе прибутку розробнику. Такі рішення є більш універсальними. задовольняють потреби тільки переважної більшості та є доволі базовими.

Також існують платформи та програми, які втілюють вузький функціонал або задовольняють спеціалізовані потреби меншості користувачів. Часто такі рішення виявляються складними для користування через штучні ускладнення інтерфейсу та неприродні рішення у сфері UX. Ці фактори відштовхують користувача і не дозволяють йому легко та швидко виконувати дії за допомогою таких платформ.

Іншими проблемами можуть бути повільність або відсутність з’єднання, помилки серверної логіки, помилки при збереженні або обробці даних. Для вирішення цих проблем у даній дипломній роботі було застосовано концепцію хмарних технологій.

Хмарні технології – це збірка концепцій, які передбачають надавання обчислювальних ресурсів, наприклад, для зберігання даних та їх обробки, без безпосереднього активного управління користувачем. Термін зазвичай використовується для опису центрів обробки даних, доступних багатьом користувачам через Інтернет.

Забезпечення послуг хмарних технологій відбувається за наявності мереж високої пропускної здатності, недорогих комп'ютерів та пристроїв зберігання даних . Важливим для хмарних технологій є також широке впровадження апаратної віртуалізації, орієнтованої на сервісну архітектуру та автономних обчислень.

Об`єктом дослідження є проектування, розробка та імплементація мобільного медичного сервісу для видаленої взаємодії користувача із державними медичними закладами з використанням хмарних технологій.

Предметом дослідження виступає мобільний медичний сервіс для видаленої взаємодії користувача із державними медичними закладами з використанням хмарних технологій.

Метою дипломного проекту є розробка та впровадження мобільного медичного сервісу для видаленої взаємодії користувача із державними медичними закладами з використанням хмарних технологій для полегшення взаємодії пацієнтів з клініками та лікувальними закладами, прискорення доставки повідомлень та загального покращення процесу роботи медичних закладів.

Для досягнення мети даної дипломної роботи необхідно виконати ряд завдань:

1. Провести передпроектне дослідження предметної області та провести огляд та аналіз аналогічних сервісів.
2. Провести дослідження мобільних та хмарних технологій з подальшим вибором визначених інструментів для розробки.
3. Обґрунтувати вибір засобів проектування та розробки платформи.
4. Розробити архітектуру та інтерфейс мобільного додатку.
5. Розробити мобільний медичний сервіс для видаленої взаємодії користувача із державними медичними закладами з використанням хмарних технологій.
6. Провести тестування та оцінку вихідного програмного продукту.

Розробка програмного продукту має цільове спрямування на сферу надання медичних послуг. Для отримання максимального результату практики використання рекомендовано інтегрувати платформу у лікувальні та профілактичні заклади.

# 1. ПОСТАНОВКА ЗАДАЧІ

## 1.1. Мета та завдання проекту

Даний проект розроблено з метою покращення зв’язку між пацієнтами клінік та лікувально-профілактичними закладами. Головним завданням є розробка сервісу, який відповідає основним вимогам для таких користувачів, забезпечує інформацією, необхідною для роботи клінік і навпаки, спрощує документообіг та пришвидшує обмін інформацією.

Таким чином, головними задачами такої платформи є:

- формування системи надання медичних послуг з можливістю відправки та прийому повідомлень як зі сторони пацієнта, так і зі сторони медичного закладу;
- спрощення процедури запису та уніфікування процесу повідомлень пацієнтів про послуги медичного закладу;
- пришвидшення роботи медичного закладу шляхом спрощення та оцифровування документообігу;
- побудова стійкого інтересу пацієнтів до послуг медичних закладів шляхом спрощення процедури спілкування з закладом.

Вчасні та достовірні відомості є ціллю роботи системи, які є необхідними для забезпечення швидкого та безперебійного обміну інформацією. Під час розв’язання цієї задачі управління інформацією здійснюється за допомогою *JSON*-об'єктів.

Для роботи система використовує такі вхідні поля та масиви даних:

- персональні дані користувачів: масив даних, який містить особисті дані та характеристики користувача;
- роль у системі: для надання спеціалізованих функцій;
- список послуг: масив, який характеризує пацієнта або клініку;
- додаткова інформація: певний масив, що затребуваний аналітичним центром.

На рис. 1.1 зображено взаємозв’язок між різними інформаційними блоками в системі.


Рисунок 1.1 Даталогічна модель задачі

Отже, основними вхідними даними для взаємодії користувача із платформою виступають: масиви персональних даних, які містять ідентифікаційні дані користувача (прізвище, ім’я, по батькові; дата народження; місце, де перебуває учасник платформи); масив, що має функції тригеру та визначає роль у системи та доступні можливості учасника платформи; масив, що містить дані про використані або пропоновані послуги; також система визначає певні додаткові дані за необхідністю (при верифікації викладача і підтвердженні відсутності спаму) та оброблює певні буферні дані, які виникають при розриві зв’язку з мережею Інтернет, задля збереження введених даних користувачем.

В таблиці 1.1 наведено перелік та опис вхідних повідомлень.

Таблиця 1.1 Класифікація вхідних повідомлень

| № | Назва | ID | Форма | Період | Термін |
|---|---|---|---|---|---|
| 1 | Персональні дані користувача | PERS | Системне повідомлення pushup | За потреби | За потреби |
| 2 | Роль користувача в системі | ROLE | Повідомлення в додатку | При надсиланні повідомлення | Без терміну |
| 3 | Список послуг клініки | SERV | Список з даними | За запитом | За потреби |
| 4 | Додаткові дані | OTH | Список у додатку | За запитом | За потреби |

Вихідні повідомлення – це такі масиви інформації, які надходять до користувача від системи. Відображення цих даних відбувається за допомогою мобільного додатку. Як видно з даталогічної моделі, кластер вихідних повідомлень складається з чотирьох масивів

При користуванні системою користувач може отримувати деяку інформацію як відповідь на свої запити. У таблиці 1.2 наведено перелік та опис вихідних повідомлень.

Таблиця 1.2 Класифікація вихідних повідомлень

| № | Назва | ID | Форма | Період | Термін |
|---|---|---|---|---|---|
| 1 | Системне повідомлення | PUNOT | Системне повідомлення pushup | За потреби | За потреби |
| 2 | Повідомлення від користувача | SMS | Повідомлення в додатку | При надсиланні повідомлення | Без терміну |
| 3 | Список користувачів зі статистикою | LIST | Список з даними | За запитом | За потреби |
| 4 | Список послуг | NAV | Список у додатку | За запитом | За потреби |

Головна задача мобільного додатку полягає в якісному та безпомилковому обміні масивами даних від сервера до клієнта та навпаки.

## 1.2. Вимоги до програмних засобів

Вимоги до програмного забезпечення – це сфера в рамках програмної інженерії, яка займається встановленням потреб зацікавлених сторін, які повинні вирішуватися програмним забезпеченням. Вимоги описують умову або спроможність, яку повинна задовольняти система або компонент системи, щоб задовольнити контракт, стандарт, специфікацію чи інший офіційно накладений документ [1].

Діяльність, пов’язана з роботою з вимогами до програмного забезпечення, може бути розбита в основному на чотири розділи:

1. Визначення – це збір та виявлення вимог зацікавлених сторін та інших джерел. Можуть бути використані різноманітні методики, такі як сесії спільного проектування заявок, інтерв'ю, аналіз документів, фокус-групи тощо.
2. Аналіз - це логічний етап, що випливає з попереднього. Аналіз передбачає досягнення багатшого та точнішого розуміння кожної вимоги та представлення наборів вимог декількома взаємодоповнюючими способами.
3. Специфікація передбачає подання та зберігання зібраних знань про вимоги на постійній та організованій основі, що полегшує ефективне спілкування та управління змінами.
4. Перевірка включає методи підтвердження того, що правильний набір вимог був визначений для створення рішення, яке задовольняє бізнес-цілі проекту.

Для визначення базових вимог до програмного забезпечення було проведено анонімне опитування за допомогою Google Forms – сервісу, який дозволяє розробляти та розповсюджувати анкети для проведення опитувань. Форма опитування наведена в Додатку 4. Головними питаннями опитування були питання, пов’язані з загальною характеристикою користувацького інтерфейсу, функціоналом, необхідним для користувачів, які є пацієнтами клінік та потребують зручного способу зв’язку з лікарем або з адміністрацією клініки. Нижче наведено графіки, які було побудовано за результатами опитувань.

На рисунку 1.2 зображено три діаграми, які описують загальні відомості про групу анонімних опитуваних. Дані охоплюють вік, стать та рід занять.

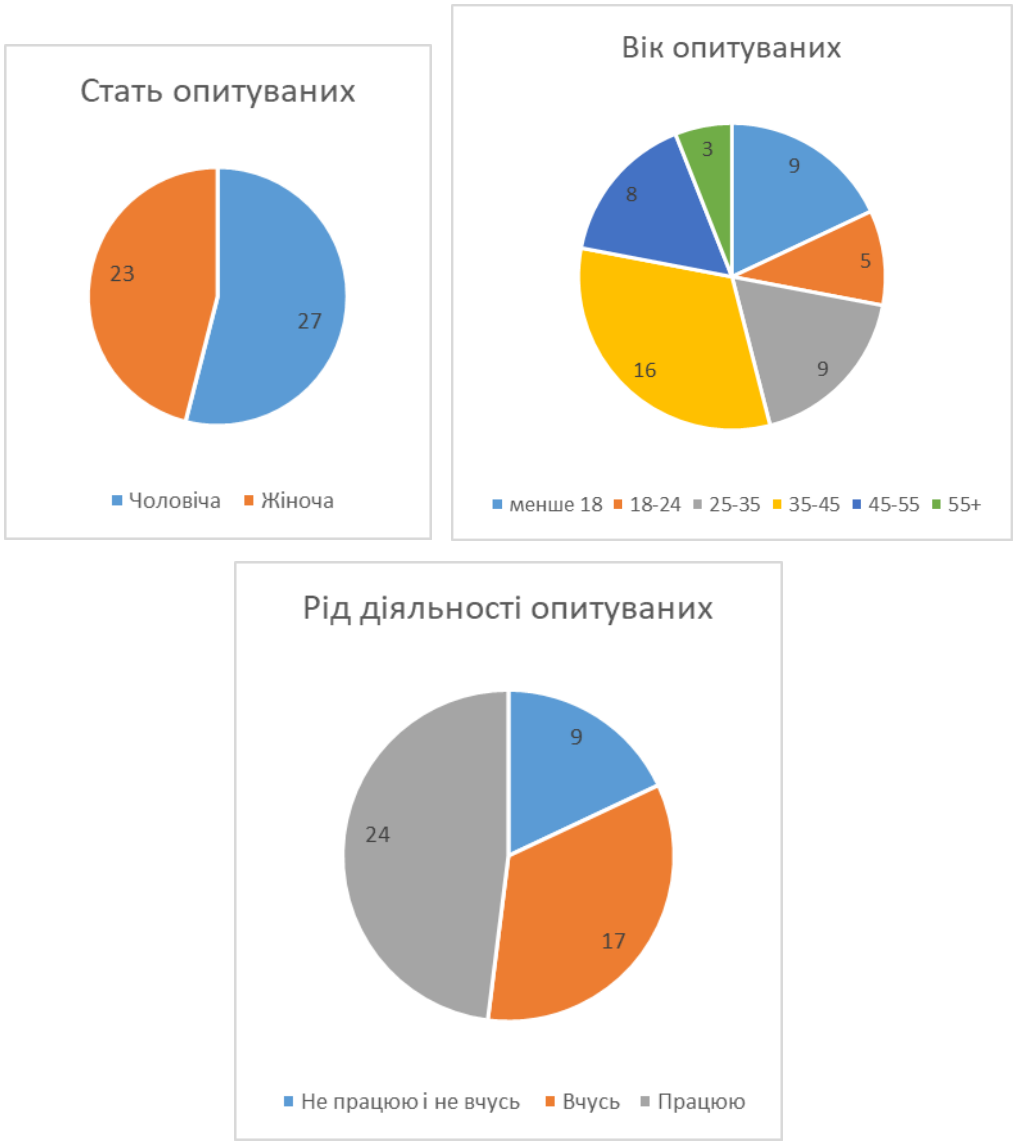



Рисунок 1.2 Діаграми 1, 2 і 3

Як видно з діаграми 1, у опитуванні прийняли участь жінки та чоловіки у майже рівному співвідношенні. Діаграма 2 показує, що найбільше опитуваних було віком 35-45 – це цільовий вік пацієнтів клінік, один з найчастіших.

На діаграмі 3 видно, що майже половина всіх опитуваних працює, і ще більше половини працюють. Для таких людей швидкий обмін інформацією є важливим та необхідним, оскільки в них є обмежена кількість часу на спілкування та візити.

На рисунку 1.3 відображено діаграму, яка показує частоту відвідувань поліклініки.


Рисунок 1.3 Частота відвідування поліклініки

Як видно на рис. 1.3, найчастіше опитувані відвідують поліклініку тільки за потреби, наприклад, при хворобі. Це є задовільним результатом, оскільки за загальними рекомендаціями необхідно відвідувати лікарню щороку, щоб слідкувати за динамікою.

На рис. 1.4 відображено співвідношення відповідей, що стосуються проблем комунікації. Майже всі стикалися з такими проблемами з різною частотою.

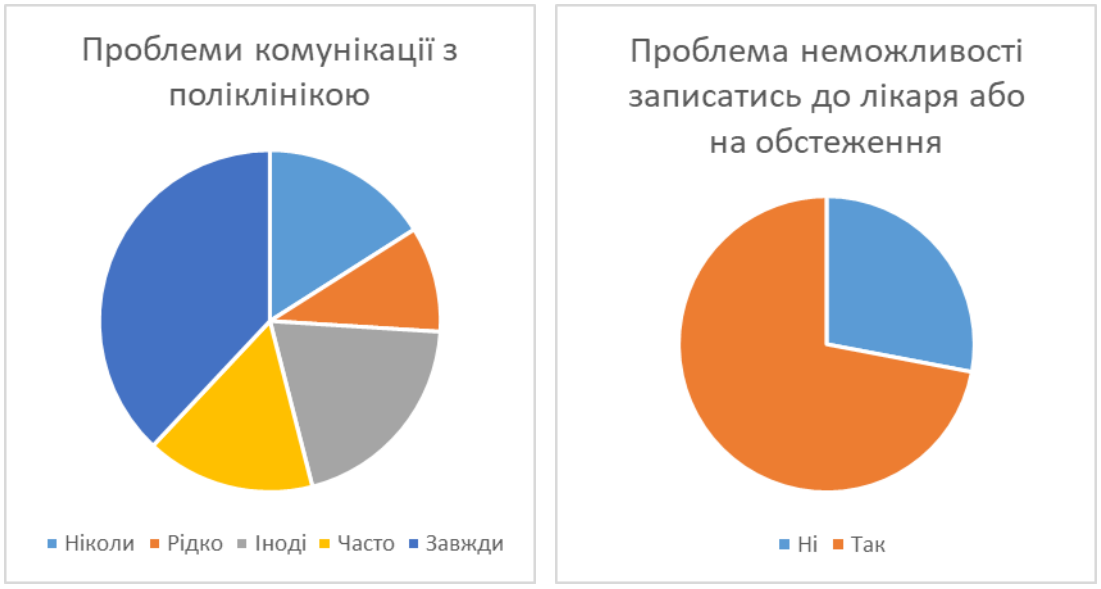



Рисунок 1.4 Проблеми поліклінік

Друга діаграма на рис. 1.4 показує, що майже три чверті опитуваних не могли записатись до лікаря або на обстеження з деяких причин. Загальний висновок полягає в тому, що більша частина опитуваних не відвідує поліклініку через проблеми комунікації або проблеми запису до лікаря. Таким чином, загальний рівень здоров'я та можливості отримати кваліфіковану медичну допомогу падає.

Аналіз опитування довів, що основними вимогами користувачів до програмного продукту є такі:

1. Надання можливості комунікації з поліклінікою регламентованим та задокументованим способом.
2. Надання можливості комунікації з власним лікарем, при потребі.
3. Надання можливості отримувати та переглядати результати обстежень або візитів до лікаря.
4. Надання можливості запису на прийом до лікаря або на обстеження.

На рис. 1.5 відображено результати відповідей на питання про концепцію мобільного додатку.


Рисунок 1.5 Концепція додатку

Результати опитування показали, що користувачі бажають отримати компактний програмний засіб з необхідним функціоналом, який не буде перевантажений непотрібними функціями.

Основною задачею розроблюваного програмного засобу є встановлення стійкого та безпечного зв’язку з базою даних, відправка та отримання даних, запитуваних користувачем [2].

Основними вимогами до бази даних є стійкість з’єднання, безпека даних та швидке реагування. Обрана система збереження та обробки даних *Firebase Realtime Database* дозволяє зберігати дані в зручному для обробки форматі, надає актуальні дані відповідно до запиту користувача та є хмарною, що економить сховище на локальному пристрої та дозволяє отримати доступ до даних з будь-якого пристрою.

Вимоги до інтерфейсу обмежуються концепцією *Material Design* – підходом, який визначає правила оформлення інтерфейсу та правила побудови логіки мобільного додатку. Інтерфейс повинен відповідати сучасним стандартам дизайну, а саме: мати єдину колірну гаму, з застосуванням головних та побічних кольорів; реалізовувати класичні компоненти інтерфейсу з достатньою нативністю; загальна композиція повинна бути простою та зрозумілою; неоднозначність компонентів повинна бути відсутня [3].

Вимоги до роботи додатку стосуються стабільності виконання задач, таких як збір, надсилання, отримання та відображення інформації. Необхідно забезпечити достатній рівень стабільності, який виражається у відсутності помилок при введенні будь-яких даних.

Перевірка на правильність введення повинна бути реалізована всередині додатку, оскільки база даних не вводить обмежень на формат даних, збережених в вузлах. Дані повинні зберігатись в базі даних, але по запиту повинно відбутись відображення необхідних даних в рамках інтерфейсу мобільного додатку.

Безпека даних забезпечується сервісом *Firebase Relatime Database Security Rules*, що виключає можливість пошкодження даних, які зберігаються в базі даних.

Кінцевий розмір додатку не повинен перевищувати 3 *Mb*, з урахуванням всіх елементів інтерфейсу та програмної складової. Кінцевий продукт повинен бути стабільним та оптимізованим для системи.

## Висновки до розділу 1

В розділі 1 було визначено мету та завдання, які необхідно вирішити в рамках даної дипломної роботи. Було сформовано основну ціль розроблюваної системи, яка полягає в забезпеченні інформацією, необхідною для роботи клінік і спрощенні документообігу, а також пришвидшення обміну інформацією.

Було розроблено даталогічну модель задачі. Ця діаграма наочно показує, яким чином відбувається обмін інформацією у вигляді масивів даних всередині складної системи. За допомогою даної моделі було виділено основні масиви, які виконують роль пакетів, які безпечно та швидко транспортуються до сховища даних та навпаки.

Вимоги до програмних засобів було сформовано та задокументовано. Було вивчено процес діяльності, пов’язаною з роботою з вимогами до програмного забезпечення. Вона була поділена на чотири етапи, кожен з яких було описано та пройдено.

На етапі визначення вимог було використано спосіб анонімного опитування як основний ресурс постачання вимог від користувачів. Темою опитування були проблеми, з якими стикаються пацієнти лікарень при спробах відвідати лікаря або записатись на прийом та обстеження. Були сформовані наочні діаграми, які пояснюють результати опитування. На основі цих результатів були виведені основні вимоги користувачів.

На наступних етапах було оглянуто технологічні та бізнес-вимоги до програмного продукту, а також створено список вимог, які були сформовані в результаті дослідження:

Вимоги до роботи додатку стосуються стабільності виконання задач, таких як збір, надсилання, отримання та відображення інформації. Необхідно забезпечити достатній рівень стабільності, який виражається у відсутності помилок при введенні будь-яких даних.

Кінцевий розмір додатку не повинен перевищувати 3 Mb, з урахуванням всіх елементів інтерфейсу та програмної складової. Кінцевий продукт повинен бути стабільним та оптимізованим для системи.

# 2. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

## 2.1. Аналітичний огляд існуючих аналогічних програмних засобів

На сьогоднішній день розвиток технологій дозволяє встановити стійку комунікацію між різними установами за допомогою Інтернету. Так, в сфері охорони здоров’я були розроблені технології, які дозволяють пацієнту налагодити зв’язок з клінікою та отримати необхідну інформацію.

Найрозвиненішим сервісом для виклику лікаря та запису є *helsi.me*. Це всеукраїнський сервіс, який надає послуги з охорони та підтримки здоров’я. Послуги, які надає даний сервіс, такі:

- пошук лікаря за спеціальністю, прізвищем, адресою або клінікою, та запис на прийом;
- доступ до інформації пацієнта, такої як виписки та результати аналізів;
- побічний сервіс, який надає можливість замовити ліки.

Такі послуги є корисними для громадян, оскільки економлять час та дозволяють документувати та зберігати інформацію про візити та обстеження. Недоліком цієї системи є відсутність мобільного додатку, що призводить до неможливості користування системою без підготовчих заходів.

Загалом, сервіси з надання послуг охорони здоров’я не повинні бути перевантажені функціоналом, як це виконано у *helsi.me*, оскільки пацієнти не завжди володіють достатнім рівнем обізнаності, щоб не помилитись при пошуку або виклику лікаря.

Головною ціллю такого сервісу має бути простий та безпроблемний виклик лікаря або запис до нього у клініці. Сервіс *helsi.me* надає таку послугу, але пошук організований складним чином, так що людина, яка не має знань про клініку, в якій вона хоче обстежуватись, не зрозуміє, який саме спеціаліст їй потрібен.

Також цей сервіс пропонує обрати та переглянути інформацію про будь-яку партнерську клініку. Не всі клініки можна знайти в цьому списку, що є недоліком, оскільки пацієнти, які вже обстежуються в інших клініках, не зможуть користуватись сервісом.

Інтерфейс даного ресурсу перевантажений компонентами, які є занадто габаритними або неоднозначними, що знижує рівень розуміння інтерфейсу користувачем. Такий сервіс повинен бути доступним для користувача будь-якого віку та рівня освіти, тому захаращений інтерфейс є великим недоліком.

В Україні поширені також сайти клінік, кожен з яких належить окремому закладу і дозволяє записатись тільки в цей заклад. Такий підхід обмежує пацієнта у виборі, а також часто не має достатнього рівня технологій для ведення щоденника пацієнта. Це означає, що громадянину доведеться дізнаватись інформацію по телефону або ж зберігати велику кількість паперових документів, що є незручним.

В таблиці 2.1 наведено приклади таких сайтів та їх короткий опис.

Таблиця 2.1 Приклади сервісів окремих поліклінік

| № | Назва та адреса | Опис |
|---|---|---|
| 1 | Healthy and Happy https://hh.com.ua/en/ | Сайт приватної клініки, який дозволяє переглянути список послуг, лікарів, та інформацію про клініку. Не надає можливості запису до лікаря або виклику. |
| 2 | Клініка Борис http://boris.kiev.ua/ | Сайт приватної поліклініки, не має можливостей для запису та виклику лікаря, але має форму зворотного зв’язку. |
| 3 | Добробут https://www.dobrobut.com | Сайт приватної поліклініки, який надає повну інформацію про послуги, але не дозволяє запис або виклик лікаря. |

Таким чином, можна зробити висновок про те, що сервіс з надання медичних послуг повинен бути зручним, простим та повністю зрозумілим, не перевантаженим зайвими послугами та компонентами інтерфейсу.

## 2.2. Обґрунтування вибору підходів і технологій

### 2.2.1. Вибір засобів проектування архітектури програмного продукту

Архітектура програмного забезпечення стосується фундаментальних структур програмної системи та дисципліни створення таких структур і систем. Кожна структура містить програмні елементи, взаємозв’язки між ними та властивості як елементів, так і взаємозв’язків [4].

Концепція архітектури програмного забезпечення є абстракцією системи для спрощення розуміння її людиною. Такий підхід дозволяє проаналізувати очікувану поведінку системи ще на етапі проектування, без потреби впровадження. Архітектурна модель впливає на подальшу розробку, впровадження та підтримку системи, документуючи зв’язок системи з зовнішніми джерелами інформації [5].

Архітектура програмного забезпечення організовує такі характеристики:

- модульність: для простоти та надійності складну систему розділяють на модульні підсистеми з обмеженою функціональністю, які взаємодіють одне з одним. Завдання на етапі розробки архітектури полягає в виділенні точних функцій кожного модуля та організації взаємозв’язків;
- якісні характеристики системи: на цьому етапі також важливо розробити комплекс підходів, який забезпечує надійність, можливість підтримки, способи використання системи тощо. Також цей пункт включає різноманітні технічні вимоги, які поставлені в технічному завданні;
- концепція побудови системи: виділення деякого єдиного способу розробки системи, для спрощення процесу та уніфікування;
- документування функціональності системи: зазначення всіх можливостей системи та умов для їх реалізації.

Для зображення архітектури програмного забезпечення використовують архітектурні моделі – високоінформативні діаграми, головним завданням яких є ілюстрування конкретного набору рішень, притаманних структурі та дизайну системи. Архітектурні моделі є документуванням вирішень в архітектурі програмного забезпечення. Моделі повинні відповідати таким вимогам:

- висока інформативність: однієї самостійної діаграми повинно бути достатньо для опису деяких характеристик одного самостійного модуля. Метою є мінімізація інформації та її повна зрозумілість;
- стандартність: використання деяких затвердженних методологій підвищує якість та однозначність розроблених діаграм. Найчастіше використовують *UML;*
- наочність: діаграма повинна бути чіткою та ясною, без зайвої інформації та заплутаності, з мінімальною кількістю графічних елементів;
- зосередженість: відображення одного блоку функціональності або однієї реалізації на окремій діаграмі. Таким чином досягається модульність;
- ясність: розроблювана модель повинна чітко відповідати на деяке запитання, висвітлюючи тільки необхідні аспекти, які важливі для однієї локальної проблеми;
- часткова несамостійність модулів: окремі частини системи не повинні повністю самостійно реалізувати власний функціонал, оскільки вони є складовою більшого комплексу. Вони мають використовувати зовнішню інформацію і взаємодіяти з іншими компонентами програмного забезпечення;
- виділений рівень абстракції: розкривання функціоналу повинне описувати деякі рішення без поглиблених даних, які не є важливими [6].

Загалом, створення архітектурних моделей є обов’язковою складовою процесу створення програмного забезпечення. Існує декілька програмних рішень, які дозволяють графічно зображати схеми та діаграми.

Для розробки архітектурних діаграм було обрано середовище *Draw.io*, оскільки цей сервіс дозволяє використовувати хмарні технології, є простим та зрозумілим в роботі, а також надає можливість розробки діаграм будь-яких типів.

### 2.2.2. Вибір засобів проектування інтерфейсу користувача

Мобільні додатки – це програмні засоби, розроблені для мобільних пристроїв, таких як смартфони та планшети. Такі додатки мають унікальні особливості, які виділяють їх як окремий тип додатків. Мобільні платформи відрізняються компактністю, невеликим розміром зображення, обмеженими обчислювальними можливостями тощо. Таким чином, для забезпечення нативності додатку, яка проявляється зрозумілістю, простотою, ергономічністю для користувача, необхідно розробити інтерфейс, який відповідатиме особливим вимогам мобільних пристроїв [7].

Користувацький інтерфейс (*User Interface*) у галузі промислового дизайну– це середовище, де відбувається взаємодія між людиною та пристроєм. Мета цієї взаємодії полягає в тому, щоб забезпечити ефективну роботу та управління пристрою з боку людини. Для сприяння процесу роботи користувача з пристроєм, в користувацькому інтерфейсі обов’язково має бути присутній елемент зворотного зв’язку для повідомлень від пристрою для користувача. Міркування щодо дизайну пов'язані з такими дисциплінами, як ергономіка та психологія [8].

З розвитком технологій кожен розробник має забезпечити якісний графічний інтерфейс для програмного продукту (*GUI*). Будь-який якісний інтерфейс повинен відповідати вимогам таких якісних характеристик, наведених в таблиці 2.2.

Таблиця 2.2 Якісні характеристики інтерфейсу користувача

| № | Назва характеристики | Опис |
|---|---|---|
| 1 | Однозначність | Унікальність функціоналу окремих компонентів |
| 2 | Виразність | Зрозумілість функцій кожного компонента |
| 3 | Впізнаваність | Використання широко відомих символів та елементів |
| 4 | Послідовність | Логічна узгодженість груп компонентів |
| 5 | Естетичність | Відповідність елементів деяким психологічним нормам |
| 6 | Ергономічність | Вимога мінімальної кількості зусиль для користування |

Успішна розробка програмного продукту обов’язково включає в себе етап розробки інтерфейсу користувача. Графічна розробка користувацького інтерфейсу (*UI*) є ітеративною методикою розробки, яка полягає в попередній розробці макету. Основними цілями прототипування інтерфейсу є:

- визначення артефактів інтерфейсу та аналіз можливих проблем, які пов’язані з функціональною реалізацією окремих елементів;
- визначеннння загального дизайнерського рішення;
- реалізація вимог до програмного забезпечення без застосування засобів програмування та витрат;
- розуміння концепції розроблюваного програмного забезпечення;
- потенційна модель, яка лежить в основі програмного забезпечення.

Процес розробки користувацького інтерфейсу полягає в розробці окремих елементів інтерфейсу за допомогою графічних інструментів. Важливим аспектом є вибір типу графіки, який використовується для графічного інтерфейсу. На сьогоднішній час, існує два основних типи графіки: растровий та векторний.

Растровий тип графіки є зручним при побудові зображень, що складаються з пікселів – найменших одиниць зображення. Такий тип не дозволяє застосовувати масштабування, а розмір файлів досягає великих розмірів, що може вплинути на дієздатність програмного забезпечення.

Векторний тип графіки використовує математичні вирази та функції для інтерпретації графічних елементів. Такий тип переважно використовують в графічному дизайні, оскільки зображення не навантажують пристрій і є оптимізованими [9].

Проектування інтерфейсу користувача є важливим етапом, впродовж якого виявлення та виправлення помилок та неточностей, а також внесення будь-яких поправок не вимагає витрат часу, зусиль та коштів. Існує велика кількість продуктів, орієнтованих на розробку графічних елементів для різних цілей та галузей. Для проектування інтерфейсу даного програмного продукту було використано Adobe Illustrator як векторний редактор графіки.

### 2.2.3. Вибір засобів прототипування програмного продукту

Прототип – це ранній зразок деякого програмного продукту, побудований для перевірки концепції чи процесу. Це термін, який використовується в різних контекстах, включаючи семантику, дизайн та програмування програмного забезпечення. Прототип, як правило, використовується для оцінки дизайну інтерфейсу [10].

Прототипування забезпечує документування специфікацій для реальної системи. У деяких моделях робочого процесу проектування прототипу є кроком між формалізацією та оцінкою ідеї.

Для мобільних додатків обов'язковим є прототипування інтерфейсу користувача, яке засноване на розробленому дизайні інтерфейсу за допомогою векторних інструментів. На цьому етапі розроблюються такі риси додатку, які характеризують його *user experience* – здатність взаємодіяти з користувачем та надавати коректний відгук на будь-яку його дію. Користувацький досвід – це враження та ставлення людини до процесу користування мобільним додатком. Він включає практичні, досвідні, афективні, змістовні та якісні аспекти взаємодії людини та додатку.

Крім того, воно включає сприйняття людиною системних аспектів, таких як корисність, простота використання та ефективність. Користувацький досвід може носити суб'єктивний характер в тій мірі, в якій йдеться про індивідуальне сприйняття та думку стосовно товару чи системи.

Для прототипування інтерфейсу користувача необхідно розробити анімацію при взаємодії користувача з елементами інтерфейсу та загальний взаємозв'язок між екранами, які відображають різну інформацію.

На сьогоднішній день, існує велика кількість засобів для прототипування інтерфейсу. Серед них можна виділити деякі найбільш сучасні та розвинені інструменти, які відповідають вимогам для розробки прототипів користувацького інтерфейсу для мобільних додатків. Таким чином, для розробки прототипу користувацького інтерфейсу для даного проекту було обрано *Figma* та *Adobe Experience Design*.

### 2.2.4. Вибір середовищ розробки та мови програмування

Інтегроване середовище розробки (*IDE*) – це програмне забезпечення, яке надає комплексні засоби для розробки програмного забезпечення. Середовище розробки зазвичай складається щонайменше з редактора вихідного коду, засобів автоматизації побудови та налагоджувача.

Інтегровані середовища розробки розроблені для досягнення максимальної продуктивності, забезпечуючи доступ до компонентів з подібними інтерфейсами користувача. *IDE* представляють єдину програму, в якій відбувається вся розробка.

Однією з цілей *IDE* є зменшення об’єму конфігурації, необхідної для об'єднання декількох інструментів, натомість вона забезпечує той самий набір можливостей, що і один згуртований блок. Скорочення часу налаштування може збільшити продуктивність розробника, особливо у випадках, коли навчання *IDE* відбувається швидше, ніж інтеграція та вивчення всіх окремих інструментів вручну.

Більш жорстка інтеграція всіх завдань з розробки може потенційно підвищити загальну продуктивність. Наприклад, код можна безперервно аналізувати під час його редагування, забезпечуючи миттєвий зворотній зв'язок при введенні синтаксичних помилок, таким чином, дозволяючи розробникам виправляти код набагато швидше та простіше [11].

Програмні продукти для *Android* вимагають розробки не тільки безпосередньо файлів програми, але і підключення додаткових модулів, які дозволяють додатку існувати в системі.

На сьогоднішній час, існує достатньо різноманітних інструментів для розробки додатків для *Android*. Найгнучкішим інструментом є *Android Studio* – комплексне середовище з функціями графічного відображення інтерфейсу, інструментами для швидкої розробки, компілятором та віртуальною машиною. Цей інструмент було обрано для розробки програмного продукту [12].

### 2.2.5. Вибір платформи для зберігання та обробки даних

Розроблюваний мобільний додаток потребує зв’язку з сервером, оскільки це забезпечує обмін та обробку даних між різними клієнтами системи. Серверна частина системи – це комплексне програмне забезпечення, яке відповідає за віддалене збереження та обробку інформації, яка надходить від користувачів. Використання концепції «клієнт-сервер» дозволяє позбавитись від перевантаження пристрою даними, оскільки всі вони знаходяться на віддаленому носії та завантажуються при потребі за допомогою Інтернету [13].

Розробка на стороні сервера – це комплекс, який передбачає використання спеціально розроблених сценаріїв дій на сервері, тобто, на віддаленій машині, завданням яких є формування відповіді на запити, які надсилає користувач.

Серверна частина обов’язково повинна реалізовувати як безпосередньо сценарії для взаємодії з користувачем, так і сховище даних, виконане з деяким підходом та розроблене за деякою структурою, яка дозволяє систематизувати дані, які надходять та зберігаються на сервері.

Для розробки серверної частини даного проекту було обрано технологію *NoSQL*. Концепція *NoSQL* – це підхід до розробки баз даних, який дозволяє реалізувати найрізноманітніші моделі даних, включаючи формати ключових значень, документ, стовпчик та графік. *NoSQL* розшифровується як «не тільки *SQL*» і є альтернативою традиційним реляційним базам даних, в яких дані розміщуються в таблицях, а схема даних ретельно розробляється перед створенням бази даних. Бази даних *NoSQL* особливо корисні для роботи з великими наборами розподілених даних, які не мають початкової структури [14].

*NoSQL* є гнучкою технологією, яка дозволяє оновлювати дані без потреби перезавантажувати їх відображення. Така оптимізована робота досягається завдяки набору підходів, які відрізняються від традиційних концепцій.

Концепція *key-value* реалізує просту модель даних, яка поєднує унікальний ключ із пов’язаним значенням. Оскільки ця модель проста, вона дозволяє розробляти базу даних, яка складається з ключових значень. Такі значення які є надзвичайно ефективними та гнучкими для управління користувачами та великими об’ємами даних.

Бази даних, які побудовані на *NoSQL* швидко розробляються та розгортуються на сервері. Оскільки ця технологія дозволяє не використовувати структуру як основну засаду бази даних, з’являється можливість проводити запити та роботу з даними асинхронно. Класичний підхід забороняє асинхронні запити, оскільки вони можуть призвести до пошкодження даних або всієї структури бази даних [15].

Дані в таких сховищах зберігаються в спеціальних файлах, які мають просту структуру у вигляді вузлів та значень, яким вони належать (рисунок 2.1). В полях може знаходитись будь-яке значення, оскільки для даних не існує обмежень.

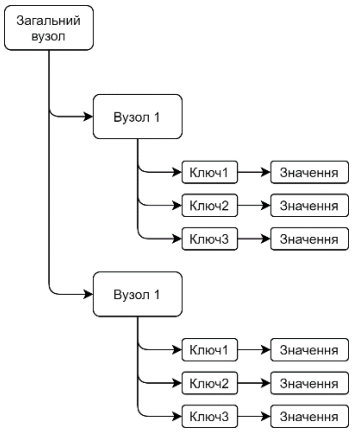



Рисунок 2.1 Схема файлу NoSQL

Ця технологія дозволяє задати шаблон для даних, або деяку схему, та запрограмувати додаток так, щоб він збирав та надсилав дані тільки в такому форматі. Кожен новий вузол буде відповідати вимогам та його структура буде такою ж, як і в попередніх вузлах. Цю перевагу використовують для оптимізації роботи додатку, узагальнюючи формат даних [16].

Для зберігання та обробки даних було використано сервіс *Firebase Realtime Database* – хмарний сервіс, який дозволяє зберігати дані в *NoSQL* та обробляти їх.

*Firebase Realtime Database* – це хмарна база даних. Дані зберігаються у форматі *JSON* і синхронізуються в режимі реального часу з кожним підключеним клієнтом. При використанні мобільних додатків всі користувачі мають можливість взаємодіяти з одним екземпляром бази даних *Realtime* і автоматично отримують оновлення з найновішими даними.

Замість типових *HTTP*-запитів *Firebase Realtime Database* використовує синхронізацію даних – щоразу, коли дані змінюються, будь-який підключений пристрій отримує це оновлення.

Додатки, що використовують *Firebase Realtime Database* залишаються чутливими до змін даних навіть у режимі відсутності доступу до інтернету, оскільки пакет *Firebase Realtime* зберігає дані на локальному диску. Після того, як з'єднання буде відновлено, клієнтський пристрій отримує всі пропущені зміни, синхронізуючи його з поточним станом сервера.

До *Firebase Realtime Database* можна отримати доступ безпосередньо з мобільного пристрою або веб-браузера; ця технологія не вимагає окремо встановленого сервера. Безпека та перевірка даних забезпечуються за допомогою *Firebase Realtime Database Security Rules* – модуль, який заснований на виразах, виконує перевірку під час читання або запису даних [17].

База даних *Firebase Realtime* дозволяє створювати додатки будь-якого призначення та розміру, надаючи безпечний доступ до бази даних безпосередньо з додатку, яким користується клієнт. Дані зберігаються на локальному рівні, і навіть без доступу до Інтернету події в реальному часі продовжують працювати, надаючи кінцевому користувачеві ті ж самі функції з деякими обмеженнями.

Коли пристрій відновлює з'єднання, база даних *Realtime* синхронізує зміни локальних даних із віддаленими оновленнями, які відбулися під час відключення клієнта, автоматично виключаючи можливість виникнення будь-яких конфліктів.

*Firebase Realtime Database* дозволяє підключатись та працювати з даними за допомогою спеціально розроблених інструментів.

## Висновки до розділу 2

У другому розділі увагу було приділено аналітичному огляду існуючих аналогічних програмних засобів. Було виділено переваги та недоліки продуктів-конкурентів, і на основі цих даних були зроблені висновки щодо функціональності та загальної організації розроблюваного продукту: сервіс з надання медичних послуг повинен бути зручним, простим та повністю зрозумілим, не перевантаженим зайвими послугами та компонентами інтерфейсу.

Також було проведено роботу з огляду та вибору засобів для проектування, розробки графіки та програмної частини продукту. Було вирішено розробити архітектуру програмного забезпечення за допомогою спеціалізованих графічних інструментів, які дозволяють будувати складні графіки та діаграми. Саме за допомогою діаграм з використанням концепції *UML* було розроблено подальші наочні зображення архітектури програмного забезпечення.

Для процесу проектування інтерфейсу користувача було виведено ряд характеристик, яким повинен відповідати розроблюваний інтерфейс. Серед них однозначність, виразність, впізнаваність, послідовність, естетичність, ергономічність. Було обрано векторний тип графіки як найоптимізованіший для мобільних пристроїв.

Прототипування продукту було вирішено проводити в спеціалізованому програмному засобі *Figma,* який дозволяє розроблювати не тільки комплексний інтерфейс для мобільного додатку, але і пов’язувати екрани та накладати анімації.

Для безпосередньо розробки програмного забезпечення було обрано інтегроване середовище для розробки *Android Studio*. Саме це середовище є оптимальним варіантом, оскільки пропонує вбудовані інструменти для розробки не тільки безпосередньо коду програми, але і графічної частини за допомогою емулятора, який здатен в режимі реального часу відтворити запрограмований інтерфейс.

# 3. ОПИС МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

## 3.1. Розробка архітектури платформи

На етапі проектування програмного засобу невід’ємною частиною є розробка архітектури додатку. Для графічного опису було обрано *UML*-діаграми.

Основною концепцією розроблюваного мобільного додатку є клієнт-серверна архітектура – тобто підхід, який дозволяє зберігати і обробляти всі дані на віддаленому пристрої. Діаграма розгортання програмного комплексу зображено на рис. 3.1.

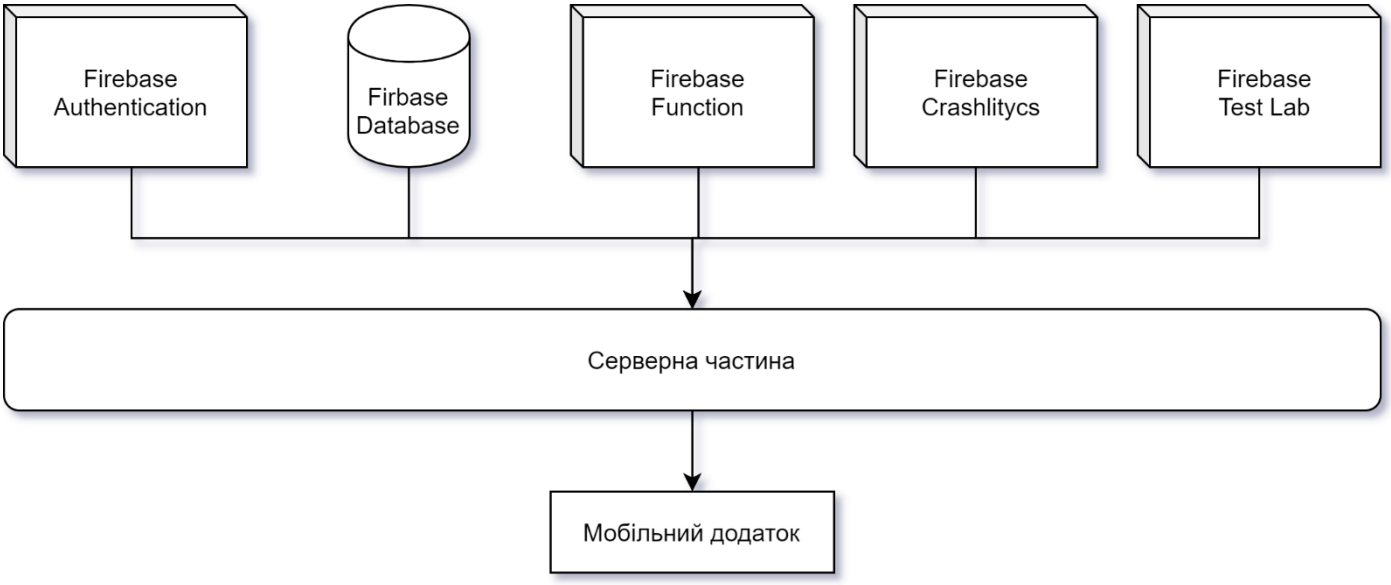



Рисунок 3.1 Діаграма розгортання «Функціональні складові платформи»

Основними компонентами системи є безпосередньо мобільний додаток та модулі платформи *Firebase*. Кожен з компонентів уособлює невід’ємну частину функціоналу, який має важливе значення для безпеки або операцій з даними.

*Firebase Realtime Database* надає послуги збереження даних, але для обробки та реалізації відповіді сервера на запити користувачів необхідний модуль *Firebase Function* – середовище для розробки та впровадження функціоналу серверу. Для збору даних користування та загальної аналітики роботи додатку підключені два модулі: *Firebase Crashlytics* та *Firebase Test Lab* [19].

Діаграма прецедентів UML – це діаграма, що відображає відносини між акторами і прецедентами і є складовою частиною моделі прецедентів, що дозволяє описати систему на концептуальному рівні. Прецедент – можливість модельованої системи (частина її функціональності), завдяки якій користувач може отримати конкретний, вимірний і потрібний йому результат. Прецедент відповідає окремому сервісу системи, визначає один з варіантів її використання і описує типовий спосіб взаємодії користувача з системою. Варіанти використання зазвичай застосовуються для специфікації зовнішнього середовища системи.

Основне призначення діаграми – опис функціональності і поведінки, що дозволяє замовнику, кінцевому користувачеві і розробнику спільно обговорювати проектовану або існуючу систему.

Для відображення моделі прецедентів на діаграмі використовуються такі символи:

- рамки системи – прямокутник з назвою у верхній частині і еліпсами (прецедентами) всередині;
- актор – символ людини, що позначає набір ролей користувача (розуміється в широкому сенсі: людина, зовнішня сутність, клас, інша система);
- прецедент – еліпс з написом, що позначає виконувані системою дії, що призводять до спостережуваних акторами результатів. Напис може бути ім'ям або описом (з точки зору акторів) того, що робить система. Ім'я прецеденту пов'язано з неперервним сценарієм – конкретної послідовністю дій, що ілюструє поведінку. В ході сценарію актори обмінюються з системою повідомленнями. Сценарій може бути приведений на діаграмі прецедентів у вигляді UML-коментаря. З одним прецедентом може бути пов'язано кілька різних сценаріїв [20].

На рис. 3.2 зображено діаграму прецедентів для розроблюваної в рамках даної курсової роботи системи.

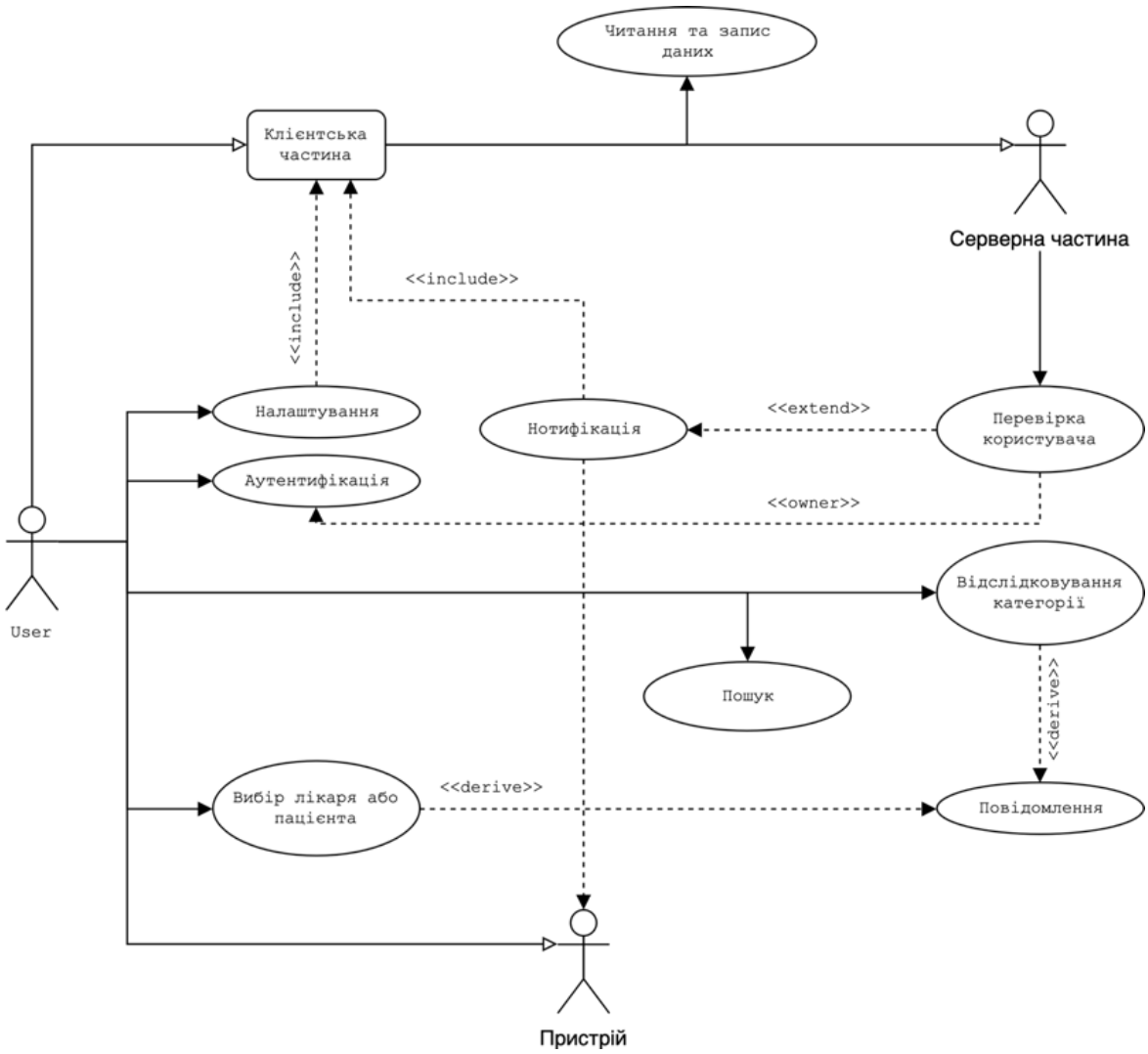



Рисунок 3.2 Діаграма прецедентів проекту

Діаграма класів – структурна діаграма мови моделювання UML, що демонструє загальну структуру ієрархії класів системи, їх кооперацій, атрибутів (полів), методів, інтерфейсів і взаємозв'язків між ними. Широко застосовується не тільки для документування та візуалізації, але також для конструювання за допомогою прямого або зворотного проектування.

Метою створення діаграми класів є графічне представлення статичної структури декларативних елементів системи (класів, типів тощо). Вона містить в собі також деякі елементи поведінки (наприклад – методи), проте їх динаміка повинна бути відображена на діаграмах інших видів (діаграмах комунікації, діаграмах станів) [21].

При поданні сутностей реального світу розробнику потрібно відобразити їх поточний стан, їх поведінку і їх взаємні відносини. На кожному етапі здійснюється абстрагування від незначних деталей і концепцій, які не належать до реальності. Класи можна розглядати з позиції різних рівнів. Як правило, їх виділяють три основних: аналітичний рівень, рівень проектування і рівень реалізації.

Діаграма класів є ключовим елементом в об'єктно-орієнтованому моделюванні. На діаграмі класи представлені в рамках, що містять три компоненти:

- У верхній частині написано ім'я класу. Імена класів починаються з великої літери.
- Посередині розташовуються поля (атрибути) класу. Вони вирівняні по лівому краю і починаються з маленької літери.
- Нижня частина містить методи класу. Вони також вирівняні по лівому краю і пишуться з малої літери.

Мова UML надає механізми для подання членів класу, наприклад атрибутів і методів, а також додаткової інформації про них. Для задання видимості членів класу (для будь-яких атрибутів або методів), ці позначення повинні бути розміщені перед ім'ям поля або методу:

- + – Публічний (Public)
- - – Приватний (Private)
- # – Захищений (Protected)

Взаємозв'язок - це особливий тип логічних відносин між сутностями, показаних на діаграмах класів і об'єктів. В UML представлені наступні види відносин:

Взаємозв’язок між об’єктами класів може бути таким:

- Залежність позначає таке відношення між класами, що зміна специфікації класу-постачальника може вплинути на роботу залежного класу, але не навпаки.
- Асоціація показує, що об'єкти однієї сутності (класу) пов'язані з об'єктами іншої сутності таким чином, що можна переміщатися від об'єктів одного класу до іншого.

• Агрегація – це різновид асоціації при відношенні між цілим і його частинами. Одне відношення агрегації не може включати більше двох класів.

• Композиція – більш суворий варіант агрегації. Відома також як агрегація за значенням.

Діаграма класів проекту зображена на рис. 3.3.

З діаграми видно, яким саме чином побудовані класи, які відображають сутності, а також відношення між класами в програмному продукті.

Так, сутність Лікар має поля, які описують особу, та методи, які дозволяють управляти обліковими записами клієнтів. Сутність Пацієнт має схожу структуру, оскільки також може управляти обліковим записом та залишати повідомлення. Сутності Карта Пацієнта, Захворювання, Повідомлення та Прийом мають тільки поля для зберігання даних.

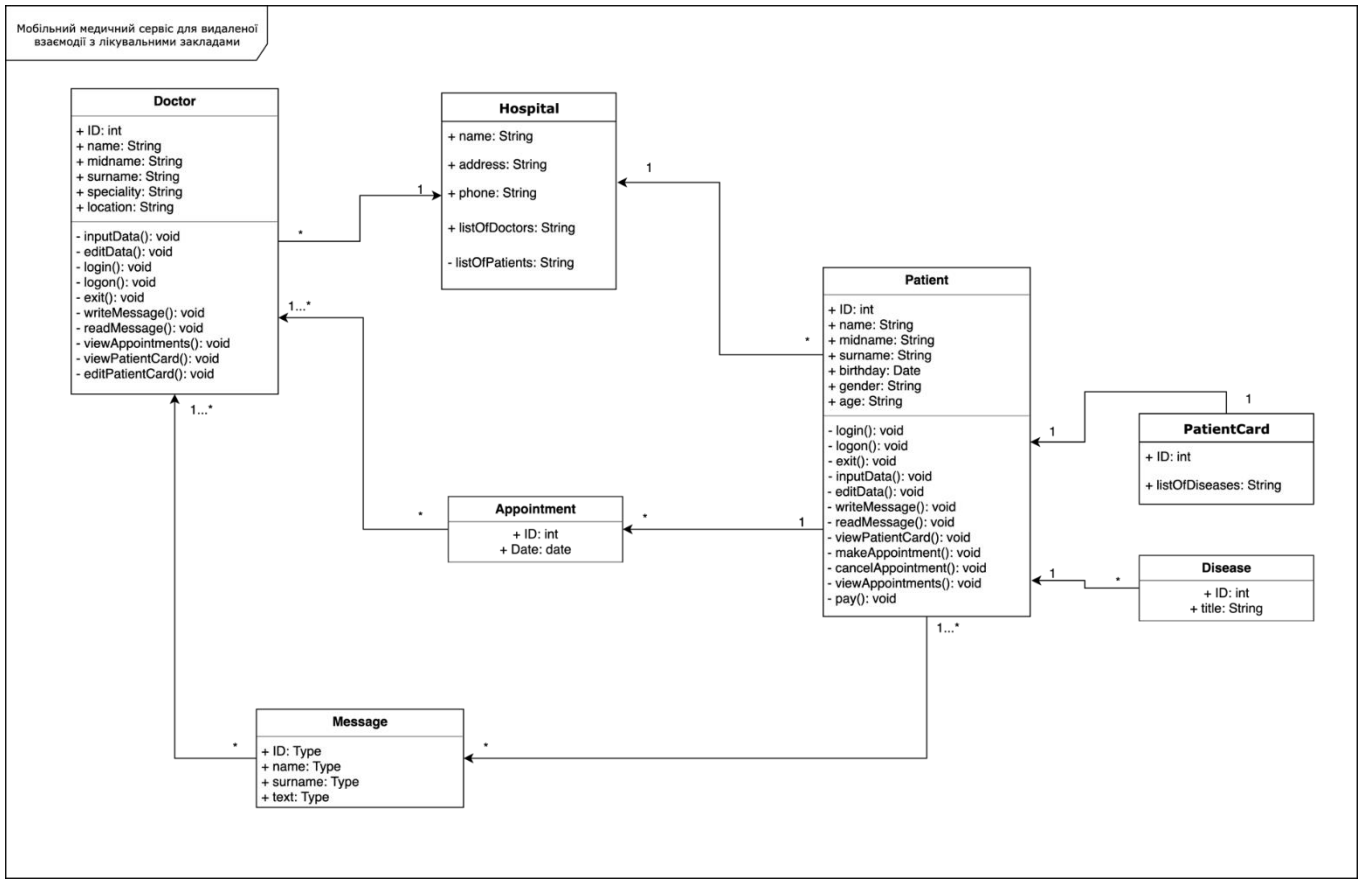



Рисунок 3.3 Діаграма класів проекту

## 3.2. Опис процесів функціонування платформи

Розроблювана система є системою, компоненти якої поділені за функціональністю та розробляються як окремі модулі. Головною задачею функціонування платформи є обробка інформації та забезпечення обміну масивами даних серверу з мобільним додатком.

Дані, які надходять у систему в формі масивів, використовуються для категоризації, надання послуг та спілкування між користувачами. Перелік масивів, використовуваних під час роботи, подано в таблиці 3.1.

Таблиця 3.1 Перелік використаних масивів

| № | Масив | Ідентифікатор | Максимальна кількість записів, рядок |
|---|---|---|---|
| 1. | Нотифікація від системи | *PUNF* | 400 |
| 2. | Повідомлення від учасника | *SMS* | 200 |
| 3. | Список учасників зі статистикою | *LIST* | 200 |
| 4. | Відслідковуванні категорії | *NAVD* | 100 |
| 5. | Персональні дані | *PERS* | 500 |
| 6. | Роль у системі | *ROLE* | 300 |
| 7. | Карта пацієнта | *PTCD* | 100 |
| 8. | Інші дані | *OTH* | 400 |

Результати роботи сервісу використовуються для поточного контролю надавання даних до функціонального та аналітичного блоку платформи, які в свою чергу формують відправку нотифікацій, переклад та форматування повідомлень, формування списків учасників для огляду та моніторинг.

Таким чином, процесами є саме функції обміну, збору та обробки інформації, яка в свою чергу є структурною одиницею у вигляді масиву. Побічним функціоналом є аутентифікація користувача та системні процеси, які збирають статистику користувача.

Процес аутентифікації з точки зору системної реалізації зображено на рис. 3.4 у вигляді діаграми діяльності. Верифікація користувача проходить в чотири етапи: дії користувача, програмний продукт, аналітичний центр та модуль аутентифікації.

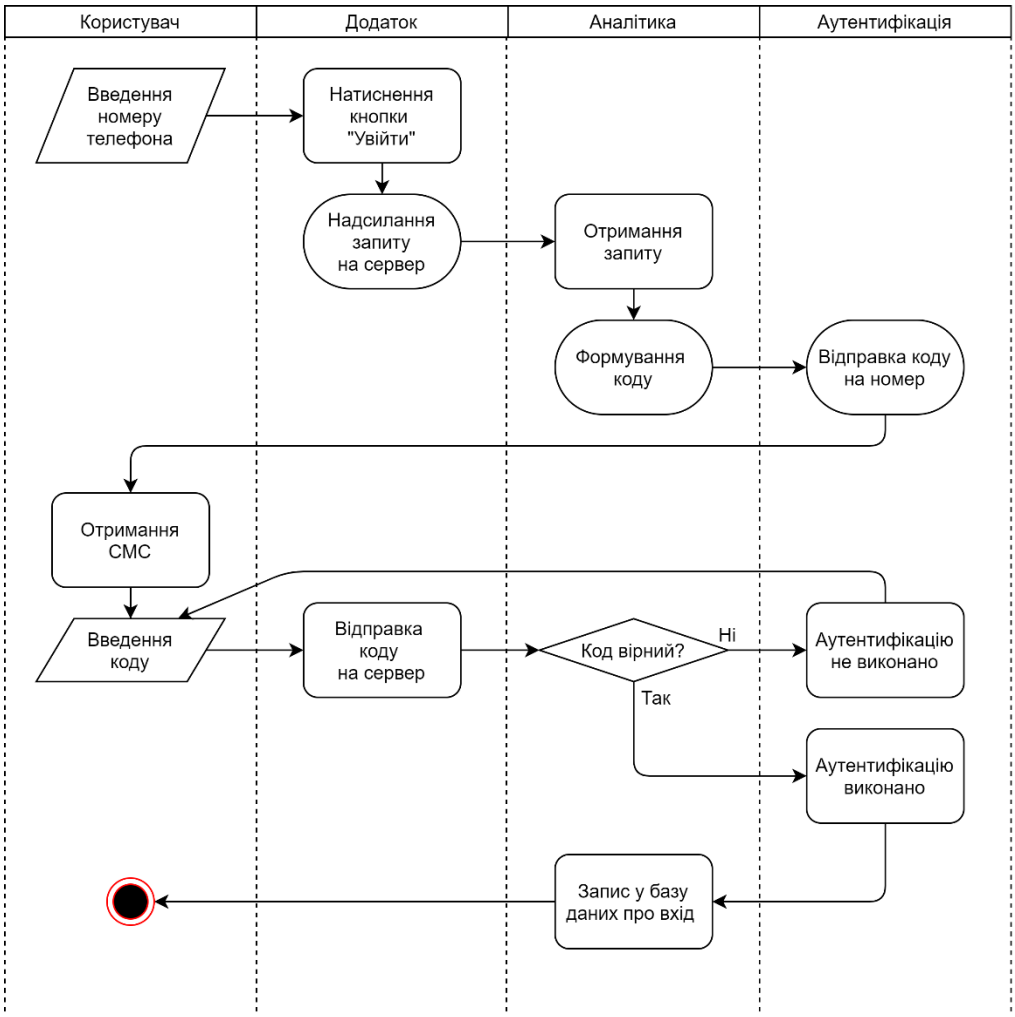



Рисунок 3.4 Діаграма діяльності «Процес аутентифікації користувача»

Модуль аналітики відіграє важливу роль: відслідковування ботів, блокування спаму, перевірки на наявність нецензурних слів, підставні акаунти. Але основною задачею для модулю є перший запуск програмного продукту користувачем. Після виклику додаткових компонентів серверної частини, аналітичний блок надає команди сформувати SMS-код, перевірити його введення, записати в базу номер телефону, пристрій та статус користувача. Отримавши підтвердження попередніх стадій, аналітичний блок формує певний тимчасовий ідентифікатор користувача.

Повідомлення користувачів про події всередині додатку відбуваються за участі аналітичного центу та безпосередньо бази даних. На рис. 3.5 відображено процес надсилання та отримання нотифікації.

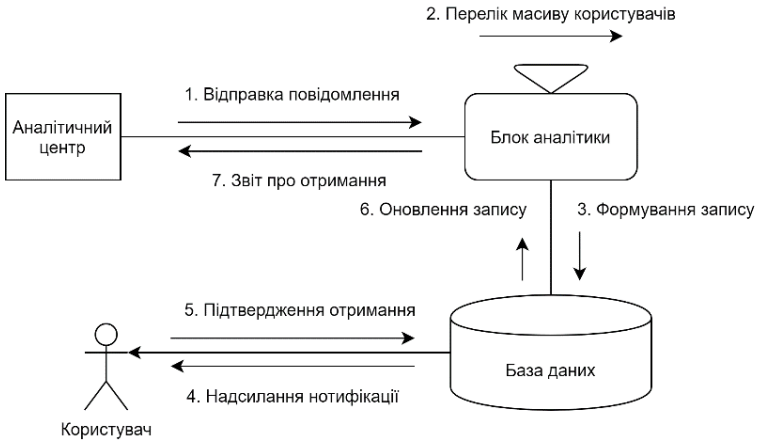



Рисунок 3.4 Діаграма кооперації «Відправка нотифікації системою»

Ключовою функцією мобільного додатку користувача є можливість надсилання масивів інформації на сервер. Для цього користувач повинен ввести дані, які запитуються сервером або додатком, та підтвердити відправку. На рис. 3.5 відображено життєвий цикл такого масиву.

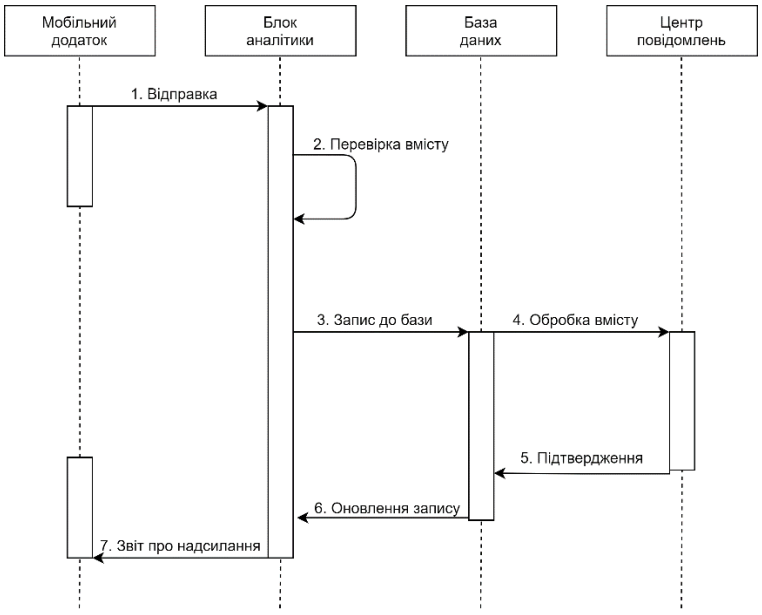



Рисунок 3.5 Діаграма послідовності «Життєвий цикл повідомлення»

## 3.3. Розробка фірмового стилю програмного продукту

На цьому етапі головним завданням є вироблення деякого фірмового стилю додатку. Фірмовий стиль – це набір деяких характеристик, які використовуються в дизайні для того, щоб створити унікальний продукт, який відрізняється від інших продуктів та має унікальні характеристики. Розробка фірмового стилю для мобільного додатку включає розробку логотипу додатку, його фірмових кольорів, фірмових форм (якщо такі є). Розробка фірмових форм та компонентів не є обов'язковою, якщо для додатку було обрано деякий стиль, наприклад, Material [22].

На рис. 3.6 наведено основні кольори та логотип для додатку.

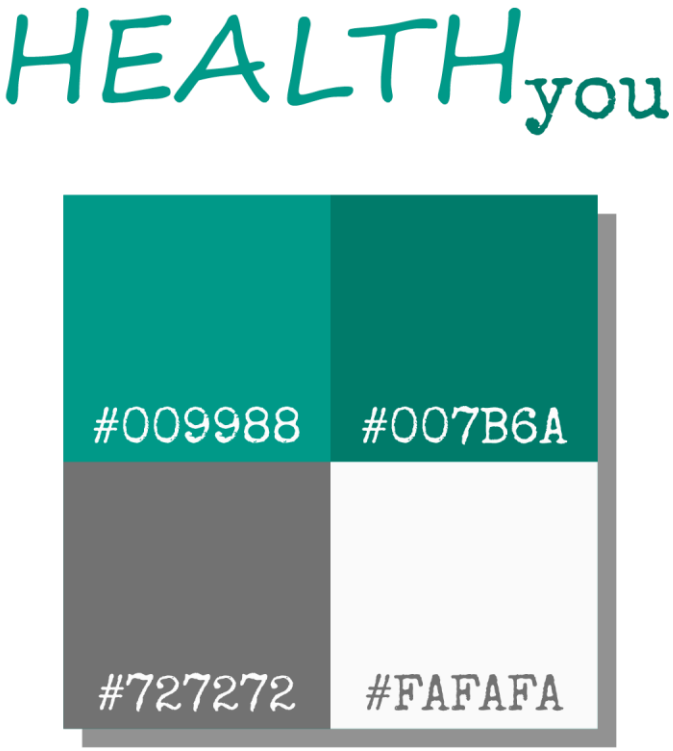



Рисунок 3.6 Фірмовий стиль розроблюваного додатку

Графічний інтерфейс користувача відіграє важливу роль у розробці програмного продукту. Інтерфейс є одним з головних факторів, якими керується користувач при формуванні свого враження від користування додатком.

Основними факторами, які впливають на якість інтерфейсу, є його простота та нативність. Обрана концепція *Material* забезпечує уніфікованість стилю всіх використовуваних компонентів, тому вони є нативними та однозначними.

Інтерфейс користувача складається з окремих компонентів, кожен з яких володіє унікальними особливостями та призначений для окремих цілей. Основними компонентами є такі:

- поля вводу даних (*EditText*);
- кнопки (*Button, FUB, SwitchButton, Checkbox, RadioButton*);
- навігаційна панель (*Navigation Drawer*);
- навігаційна стрічка (*Bottom Navigation*);
- картки представлення (*Cards*, *Chips*);
- списки (*ListView*, *RecyclerView*);
- діалогові вікна (*Dialogs*)
- меню (*Menus*).

В концепції *Material* передбачена наявність явища, яке притаманне будь-якому елементу інтерфейсу: стан компонента. Зрозумілість компонентів забезпечується їх відповіддю на дії користувача. Такими можуть бути різноманітні реакції: підсвічування, активність, неактивність, підказка тощо.

Інтерфейс користувача розробляється попередньо до прототипу, оскільки прототип базується саме на елементах інтерфейсу. Вони розробляються в векторному форматі в спеціалізованому графічному середовищі.

Для розповсюдження додатку користувачам, існує спеціальний програмний продукт, який збирає всю інформацію про додаток та маркетингові матеріали та публікує додаток в магазині *Play Market*, звідки його можна завантажити всім користувачам.

На рис. 3.7 та рис. 3.8 зображено розроблені маркетингові матеріали для додатку. Загальний набір включає іконку додатку, яка повинна бути добре впізнаваною та виконаною згідно фірмового стилю додатку, зображення, яке з’явиться в магазині з цим додатком, широке зображення для реклами та набір зображень, які є макетами для користувача.

В додатку 5 наведено зображення інтерфейсу більшого масштабу.


Рисунок 3.7 Маркетингові матеріали

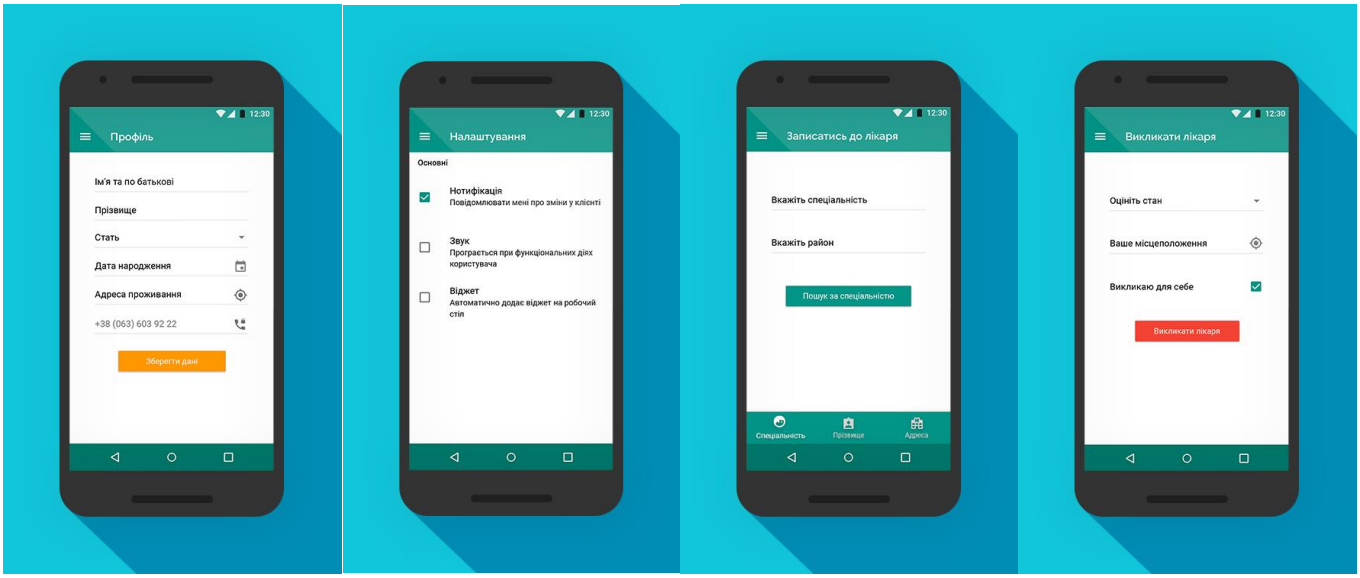



Рисунок. 3.8 Макети додатку

## 3.4. Розробка прототипу програмного продукту

Прототипування є невід'ємною частиною процесу розробки інтерфейсу, оскільки саме прототип є основою для побудови програмно реалізованого інтерфейсу. Прототип будується за допомогою спеціалізованого програмного забезпечення, яке дозволяє не тільки застосувати компоненти інтерфейсу, але і пов'язати їх логічним чином та застосувати анімацію при потребі.

Переходи та взаємодії між компонентами та вікнами реалізуються програмно, за допомогою станів додатку та екранів, а також станів компонентів. Нижче на рис. 3.9 наведено загальну схему взаємозв'язків між екранами. Схема в більшому масштабі наведена у Додатку 6.

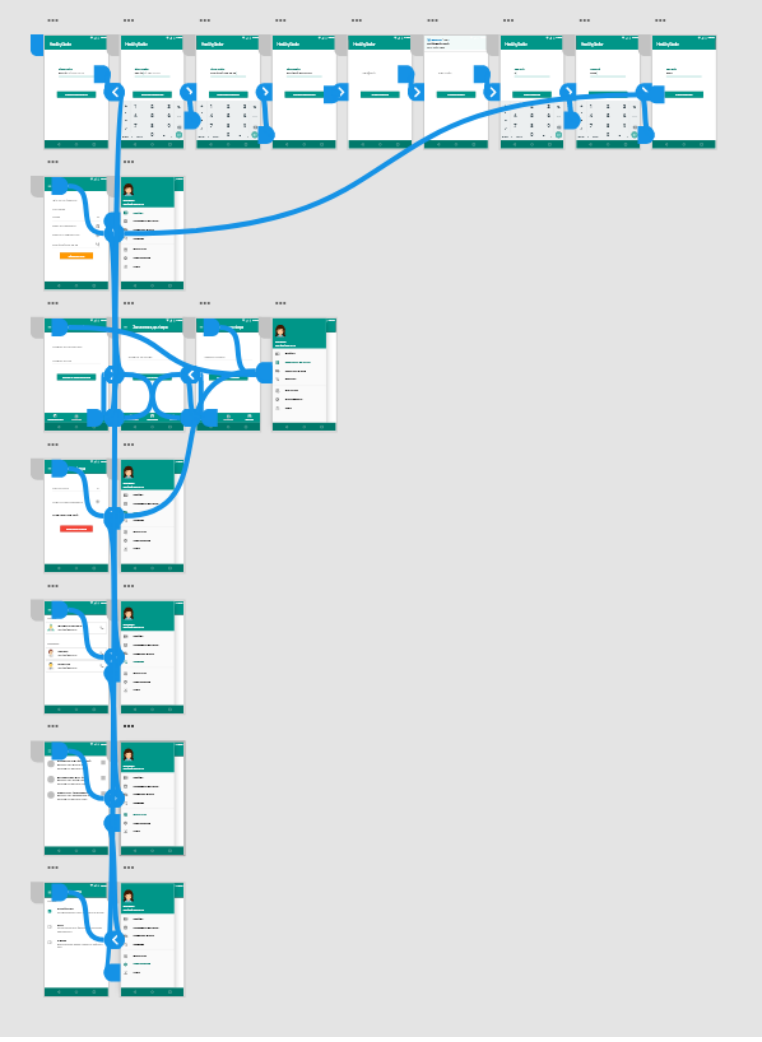


Рисунок 3.9 Загальний вигляд прототипу додатку

## 3.5. Тестування та оцінка розробленого продукту

Тестування програмного забезпечення – це процес розробленого дослідження програмного забезпечення з метою отримання інформації про якість продукту. Також часто під тестуванням розуміють процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, здійснюваний шляхом спостереження за роботою продукту в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином [23].

У широкому сенсі, тестування – це одна з технік контролю якості, яка включає планування, складання тестів, безпосередньо виконання тестування і аналіз отриманих результатів.

Існує декілька рівнів тестування, які відповідно дозволяють перевірити окремий функціонал компонентів продукту, функціонал цілих програмних модулів або функціонал продукту загалом.

Тестування компонентів – процес, в якому тестується мінімально можливий для тестування компонент, наприклад, окремий клас або функція. Часто тестування компонентів здійснюється розробниками програмного забезпечення. Яскравими прикладами такого тестування є тестування чорного ящика та тестування білого ящика.

Інтеграційне тестування – тестування, під час якого тестуються взаємодії між компонентами, підсистемами або системами. При наявності резерву часу на даній стадії тестування ведеться ітераційно, з поступовим підключенням наступних підсистем. Прикладом такого типу є модульне тестування або юніт тест [24].

Альфа-тестування – реальна робота з системою потенційними користувачами або тестувальниками. Найчастіше альфа-тестування проводиться на ранній стадії розробки продукту, але в деяких випадках може застосовуватися для закінченого продукту в якості внутрішнього приймального тестування.

Іноді альфа-тестування виконується з використанням оточення для розробки, яке допомагає швидко виявляти знайдені помилки.

Бета-тестування – тестування великою групою користувачів з метою, щоб переконатися, що продукт не містить критичних помилок. Іноді бета-тестування виконується для того, щоб отримати зворотній зв'язок про продукт від його майбутніх користувачів.

Для проекту було обрано тестування білого ящика, чорного ящика та компонентне тестування. На етапі альфа-тестування було проведено тести за допомогою управляючих графів.

Тестування білого ящика – це метод тестування програмного забезпечення, в якому внутрішня структура або реалізація деякого компонента відома. Тут визначаються спеціалізовані масиви вхідної та вихідної інформації, беручи до уваги структуру модуля [25].

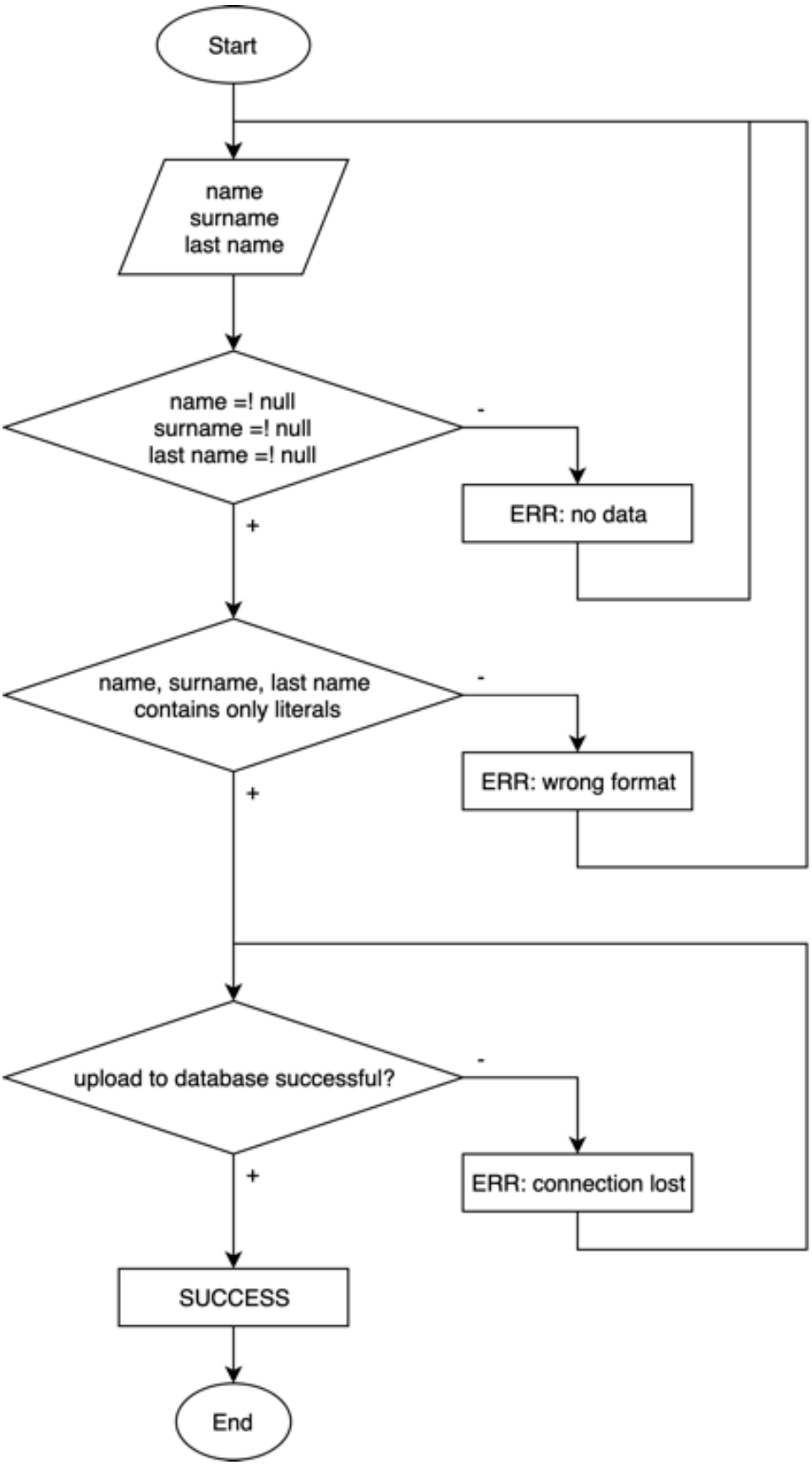
Для тестування методом білого ящика було обрано модуль введення первинних даних користувача (прізвища, імені та по-батькові) при реєстрації. Цей модуль повинен прийняти два масиви даних і обробити їх без помилок, щоб ці дані можна було безпечно відправити до бази даних.

Так, в даному проекті первинними даними можуть бути масиви, які наведено в таблиці 3.2:

Таблиця 3.2 Перелік і опис вхідних масивів

| № | Назва повідомлення | Формат | Кількість знаків |
|---|---|---|---|
| 1. | Прізвище, ім’я, по-батькові | Рядок | До 100 |
| 2. | Вік | Число | MAXINT |

Таким чином, для перевірки модулю введення первинних даних користувача (тобто, даних, які надаються при реєстрації користувача). Функція реєстрації та передачі первинних даних працює таким чином (Діаграма 2.11):




Діаграма 2.11 Алгоритм роботи програми введення первинних даних

Таким чином, алгоритм містить щонайменше 2 підмодулі, які необхідно протестувати методом білого ящика. Відомо, що модуль перевірки трьох рядків на вміст складається з двох функціональних частин: перевірка на пустоту рядка і перевірка на правильність введення (повне ім’я повинне складатись тільки з літер). Модуль відправлення даних на сервер складається з підфункції встановлення зв’язку з базою даних та підфункції відправлення даних.

Таким чином, перевірити даний алгоритм методом білого ящика можна таким чином:

Введення пустих рядків з метою перевірити підфункцію відслідковування помилки (*ERR: no data*). Очікуваною подією буде повідомлення від системи про помилку пустого рядка та повторний запит даних.

Введення рядків з числами та знаками, що не належать до множини літер. Мета: перевірити підфункцію відслідковування помилки (*ERR: wrong format*). Очікуваною подією буде повідомлення від системи про помилку невірного формату даних та повторний запит даних.

Відправлення інформації на сервер. Ця підфункція повинна встановити зв’язок та відправити дані. При виникненні помилки підфункція повинна повідомити про помилку підключення та спробувати знову. Для перевірки необхідно вимкнути інтернет-зв’язок та почати процедуру завантаження даних на сервер. Очікуваною подією буде повідомлення від системи про помилку встановлення зв’язку та повторна спроба.

Тестування чорного ящика – метод проведення тестувань, який передбачає відсутність даних про архітектуру модулю. Таким чином, тестування чорного ящика буде проводитись за допомогою порівняння введених та отриманих даних на предмет відповідності форматів або значень [26].

Для перевірки було обрано функцію аналітичного центру платформи, яка повинна відправляти сповіщення. Головним критерієм таких нотифікацій виступає зворотній звіт про успішну доставку повідомлення користувачеві.

Проводитиметься аналіз звітувань при надсиланні повідомлення, що містить до 60 символів, містить більше 60 символів та порожнє повідомлення.

Повідомлення, що має довжину більше граничного значення, поділяється на частини до 60 символів і результатом є число звітів, яке відповідає кількості частин повідомлення.

На вхід програми подається текстовий рядок: визначене повідомлення із різною кількістю символів. Результатом тестування є число звітів у Firebase Console.

Аналіз тестових кейсів підпрограми:

Вхідне повідомлення: (60 символів). Очікуваний результат: «Повідомлення надіслано: success». Одержаний результат: «Повідомлення надіслано: success». Можна стверджувати, що функція працює правильно, оскільки повідомлення вірне.

Вхідне повідомлення: (142 символи). Очікуваний результат: «Повідомлення надіслано: reports received 3». Одержаний результат: «Повідомлення надіслано: reports received 3». Можна стверджувати, що функція працює правильно, оскільки кількість пакетів відповідає очікуваній.

Вхідне повідомлення: (0 символів). Очікуваний результат: «Помилка відправки: notification is null». Одержаний результат: «Помилка відправки: notification is null». Можна стверджувати, що функція працює правильно, оскільки повідомлення вірне.

Компонентне тестування – це рівень тестування програмного забезпечення, на якому тестуються окремі одиниці або компоненти програмного забезпечення. Метою є перевірка виконання кожною одиницею програмного забезпечення відповідно до проекту. У об'єктно-орієнтованому програмуванні найменша одиниця – це метод.

Найпродуктивніше виконувати компонентне тестування, підмінюючи об’єкти класу, які наповнені невірними значеннями. Таким чином, при роботі з методом, буде зрозуміло, чи вірно побудовано алгоритм, якщо проаналізувати результати роботи методу з вірними та помилковими даними.

## Висновки до розділу 3

В даному розділі було проведено комплексну роботу з розробки архітектури платформи. Було обрано інструменти для збереження і обробки даних. Основними компонентами системи було вирішено розроблювати безпосередньо мобільний додаток та модулі платформи *Firebase*. Кожен з компонентів є частиною функціоналу, який має унікальне значення для безпеки або операцій з даними.

*Firebase Realtime Database* дозволяє виконувати збереження даних, але для обробки та реалізації дій сервера на запити користувачів було використано модуль *Firebase Function* – спеціалізоване середовище, яке надає послуги видаленого сервера.

Для збору даних про користувачів та роботу додатку, а також загальної аналітики підключені два модулі: *Firebase Crashlytics* та *Firebase Test Lab.*

Було розроблено діаграму прецедентів, яка наочно показала функціонал розроблюваного продукту, а також виділила акторів. Також було розроблено діаграму класів – схему побудови деяких визначених об’єктів та взаємозв’язків між ними.

Компоненти системи були поділені за функціональністю та розроблялись як окремі модулі. Було описано дані, які надходять до системи для обробки та збереження. Було визначено, що результати роботи сервісу використовуються для контролю обміну даними між користувачем та функціональним та аналітичним блоками платформи, які в свою чергу формують відправку нотифікацій, переклад та форматування повідомлень, формування списків учасників для огляду та моніторинг.

Було розроблено фірмовий стиль продукту, на якому базується інтерфейс та маркетингові засоби для поширення розроблюваного додатку. Було обрано фірмові кольори та розроблено логотип. На основі обраних фірмових компонентів було розроблено інтерфейс згідно вимогам та впроваджено прототип.

Було проведено оцінку розробленого продукту за допомогою тестування. Результати тестування показали достатній рівень стабільності розробленого програмного забезпечення.

# ВИСНОВКИ

Для вирішення завдань, поставлених в даній дипломній роботі, було проведено роботу з дослідження, обґрунтуванням обраних рішень, розробки, тестування та впровадження мобільного додатку.

1. Було проведено передпроектне обстеження предметної області. Висновок полягав в тому, що основним недоліком є відсутність цільової платформи. Також було проведено анонімне опитування цільової аудиторії для визначення первинних вимог.
2. Було проведено комплексне дослідження мобільних та хмарних технологій. В результаті цього дослідження були обрані технології та інструменти для застосування при розробці. Також було проаналізовано інструменти та платформи для зберігання та обробки даних
3. Було оглянуто засоби для проектування програмного засобу, такі як графічні інструменти для розробки діаграм та інструменти для розробки графічної складової. На основі оглядів було обрано гнучкі та прості в використанні інструменти для розробки архітектури.
4. Було проведено роботу з проектування та розробки архітектури проекту. Моделі та діаграми було розроблено з застосуванням спеціалізованих інструментів, для документування вимог.
5. Розробка програмного продукту проводилась у спеціалізованому середовищі, налаштованому для розробки саме мобільних додатків.
6. Проведене у два етапи тестування (розробником та кінцевим користувачем) показало відсутність критичних дефектів функціонування, продукт функціонує коректно.

Оцінка отриманого продукту підтвердила, що усі функціональні вимоги та поставлені задачі до розробки платформи було виконано.

Платформа може зайняти свою нішу у наданні послуг з реєстрації та взаємодії пацієнтів та клінік або медичних закладів і вийти у масове використання.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. McCallister J. Mobile Apps Made Simple: The Ultimate Guide to Quickly Creating, Designing and Utilizing Mobile Apps for Your Business / Jonathan McCallister. – Boston: Amazon Services LLC, 2014. – 70 с.
2. Raskin J. The Humane Interface: New Directions for Designing Interactive Systems / Jef Raskin. – Denver: Addison-Wesley Professional, 2000. – 233 с. – (1).
3. Galitz W. O. The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques / Wilbert O. Galitz. – New York: Wiley, 2007. – 888 с.
4. Documenting Software Architectures: Views and Beyond / F.Bachmann, L. Bass, D. Garlan, J. Ivers. – Boston: Addison-Wesley Professional, 2010. – 592 с.
5. Perry D. E. Foundations for the Study of Software Architecture / D. E. Perry, A. L. Wolf. – Boulder: ACM SIGSOFT, 1992. – 13 с. – (SOFTWARE ENGINEERING NOTES).
6. What is your definition of software architecture?. // Carnegie Mellon University. – 2017. – №3854. – С. 6.
7. Gero J. S. Design Prototypes: A Knowledge Representation Schema for Design / John S. Gero. – New York: AI Magazine, 1990. – 26 с.
8. van Drongelen M. Android Studio Cookbook: Design, test, and debug your apps using Android Studio / Mike van Drongelen. – Лондон: Packt Publishing, 2015. – 232 с.
9. Brewer C. Designing Interfaces / C. Brewer, J. Tidwell, A. Valencia. – New York: O'Reilly Media, 2020. – 600 с.
10. Why Is a GUI Important? [Електронний ресурс] // Reference. – 2018. – Режим доступу до ресурсу: https://www.reference.com/world-view/gui-important-95d42a64e0c41332.
11. Eckel Bruce. Thinking in Java, – Prentice Hall, 2006. – 1 видання, 1150с.
12. Meier Reto. Professional Android, – Wrox, 2018. – 4 видання, 984с.
13. Mew Kyle. Mastering Android Studio 3: Build Dynamic and Robust Android applications, – Packt Publishing, 2017. – 220с.

14. Mew Kyle. Android Design Patterns and Best Practices, – Packt Publishing, 2017. – 534с.

15. Architecture of JavaScript Applications [Електронний ресурс] // Oracle. – 2010. – Режим доступу до ресурсу: https://docs.oracle.com/cd/E19957-01/816-6411-10/getstart.htm.

16. Sullivan D. NoSQL for Mere Mortals / Dan Sullivan. – New York: Addison-Wesley Professional, 2015. – 542 с.

17. Marrs T. JSON at Work: Practical Data Integration for the Web / Tom Marrs., 2017. – 376 с.

18. Privacy and Security in Firebase [Електронний ресурс] // Firebase. – 2019. – Режим доступу до ресурсу: https://firebase.google.com/support/privacy.

19. Murphy M. The Busy Coder Guide to Advanced Android Development / Mark Murphy. – United States of America: CommonsWare, 2009. – 509 с.

20. Wiegers K. Software Requirements (Developer Best Practices) / Karl Wiegers. – Washington: Microsoft Press, 2010. – 670 с.

21. Beatty J. Software Requirements / Joy Beatty. – Washington: Microsoft Press, 2013. – 455 с.

22. Eck D. J. Introduction to Computer Graphics / David J. Eck. – New York: Hobart and William Smith Colleges, 2018. – 411 с.

23. Kaner C. Testing Computer Software / Cem Kaner. – Vancouver: Wiley, 1999. – 480 с.

24. J. Myers G. The Art of Software Testing / Glenford J. Myers. – New Jersey: John Wiley & Sons, Inc., 2004. – 375 с.

25. Crispin L. Agile Testing / L. Crispin, J. Gregory. – Toronto: Addison-Wesley, 2009. – 274 с.

26. Kaner C. Lessons Learned in Software Testing: A Context-Driven Approach / Cem Kaner. – Vancouver: Wiley, 2001. – 227 с.

# ДОДАТКИ

## Додаток А

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

## СПЕЦИФІКАЦІЯ

Листів 1

Київ – 2020

Таблиця 1 Специфікація

| Позначення | Найменування | Примітки |
|---|---|---|
| Документація | | |
| Документ | Пояснювальна записка | Даний документ вміщує інформацію щодо теоретичного та практичного підходу до розробки мобільного додатку в рамках дипломної роботи. |
| Комплекс | | |
| | | |
| | | |
| Компоненти | | |
| Мобільний додаток | HealthYou | Мобільний додаток, розроблюваний в рамках даної дипломної роботи |

Додаток Б

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

# ТЕКСТ ПРОГРАМНОГО МОДУЛЮ

Листів 1

Київ – 2020

```
import android.app.ProgressDialog;
import android.content.Context;
import android.content.Intent;
import android.graphics.Color;
import android.graphics.Typeface;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.AsyncTask;
import android.support.annotation.NonNull;
import android.support.v4.content.res.ResourcesCompat;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.RelativeLayout;
import android.widget.TextView;
import android.widget.Toast;

import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.FirebaseException;
import com.google.firebase.FirebaseTooManyRequestsException;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseAuthInvalidCredentialsException;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.auth.PhoneAuthCredential;
import com.google.firebase.auth.PhoneAuthProvider;

import java.util.concurrent.TimeUnit;

import br.com.sapereaude.maskedEditText.MaskedEditText;

public class AuthActivity extends AppCompatActivity {
    private static final String TAG = "STATUS";
    private MaskedEditText mPhoneText;
    private EditText mCodeText;

    private Button mSendBtn;
    private PhoneAuthProvider.OnVerificationStateChangedCallbacks mCallbacks;
    private String mVerificationId;
    private PhoneAuthProvider.ForceResendingToken mResendToken;

    private FirebaseAuth mAuth;

    private int btnType = 0;

    private ProgressDialog progressDialog;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_auth);
        //region [setTitle]
        TextView tv = new TextView(getApplicationContext());
```

```
        RelativeLayout.LayoutParams lp = new
RelativeLayout.LayoutParams(ActionBar.LayoutParams.WRAP_CONTENT,
ActionBar.LayoutParams.WRAP_CONTENT);
        tv.setLayoutParams(lp);
        tv.setText(R.string.app_name);
        tv.setTextSize(20);
        tv.setTextColor(Color.parseColor("#FFFFFF"));
        Typeface typeface = ResourcesCompat.getFont(getApplicationContext(),
R.font.raleway_semisold);
        tv.setTypeface(typeface);
        getSupportActionBar().setDisplayOptions(ActionBar.DISPLAY_SHOW_CUSTOM);
        getSupportActionBar().setCustomView(tv);
        //endregion

        //

        mPhoneText = (MaskedEditText) findViewById(R.id.phoneNumber);
        mCodeText = (EditText) findViewById(R.id.smsCode);
        mSendBtn = (Button) findViewById(R.id.sendNumber);
        mAuth = FirebaseAuth.getInstance();

        progressDialog = new ProgressDialog(this);
        progressDialog.setMessage("Зачекайте, код відправлено");
        progressDialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);
        //progressDialog.setIndeterminate(true);

        mSendBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                if (btnType == 0) {
                    mPhoneText.setEnabled(false);
                    mSendBtn.setEnabled(false);
                    String phone = mPhoneText.getText().toString();
                    String ip_sendstr_1 = phone.replaceAll("\\(", "");
                    String ip_sendstr_2 = ip_sendstr_1.replaceAll("\\)", "");
                    String ip_sendstr_3 = ip_sendstr_2.replaceAll("\\ ", "");
                    String phoneNumber = ip_sendstr_3;
                    PhoneAuthProvider.getInstance().verifyPhoneNumber(
                            phoneNumber,
                            60,
                            TimeUnit.SECONDS,
                            AuthActivity.this,
                            mCallbacks
                    );
                } else {

                    mSendBtn.setEnabled(false);
                    mCodeText.setVisibility(View.VISIBLE);

                    String virificationCode = mCodeText.getText().toString();

                    PhoneAuthCredential credential =
PhoneAuthProvider.getCredential(mVerificationId, virificationCode);
                    signInWithPhoneAuthCredential(credential);

                }
            }
        });
```

```
        mCallbacks = new PhoneAuthProvider.OnVerificationStateChangedCallbacks() {
            @Override
            public void onVerificationCompleted(PhoneAuthCredential
phoneAuthCredential) {
                signInWithPhoneAuthCredential(phoneAuthCredential);
            }

            @Override
            public void onVerificationFailed(FirebaseException e) {
                Toast.makeText(AuthActivity.this, "Сталася помилка при верифікації
коду, 0x3", Toast.LENGTH_SHORT).show();
                if (e instanceof FirebaseAuthInvalidCredentialsException) {
                    Toast.makeText(AuthActivity.this, "Невалідний запит, 0x31",
Toast.LENGTH_SHORT).show();
                } else if (e instanceof FirebaseTooManyRequestsException) {
                    Toast.makeText(AuthActivity.this, "Закінчились СМС-квоти, 0x32",
Toast.LENGTH_SHORT).show();
                }
            }

            @Override
            public void onCodeSent(String verificationId,
                                   PhoneAuthProvider.ForceResendingToken token) {
                // Toast.makeText(AuthActivity.this, "Зачекайте хвильку, код
відправлено", Toast.LENGTH_LONG).show();


                mVerificationId = verificationId;
                mResendToken = token;

                btnType = 1;

                mCodeText.setVisibility(View.VISIBLE);
                mSendBtn.setText("Підтвердити");
                mSendBtn.setEnabled(true);
                progressDialog.show();
                new BackgroundJob().execute();
            }
        };
        //
    }
    private void signInWithPhoneAuthCredential(PhoneAuthCredential credential) {
        mAuth.signInWithCredential(credential)
                .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
                    @Override
                    public void onComplete(@NonNull Task<AuthResult> task) {
                        if (task.isSuccessful()) {
                            // Sign in success, update UI with the signed-in user's
information
                            Log.d(TAG, "signInWithCredential:success");

                            FirebaseUser user = task.getResult().getUser();

                            Intent mainIntent = new Intent(AuthActivity.this,
UserGrantedActivity.class);
                            startActivity(mainIntent);
                            finish();
                            // ...
                        } else {
```

```
                            // Sign in failed, display a message and update the UI
                            Log.w(TAG, "signInWithCredential:failure",
task.getException());
                            Toast.makeText(AuthActivity.this, "Сталася помилка при
аутентифікації, 0x2", Toast.LENGTH_SHORT).show();
                            if (task.getException() instanceof
FirebaseAuthInvalidCredentialsException) {
                                // The verification code entered was invalid
                            }
                        }
                    }
                });
    }
    public static boolean isNetworkAvailable(Context con) {
        try {
            ConnectivityManager cm = (ConnectivityManager) con
                    .getSystemService(Context.CONNECTIVITY_SERVICE);
            NetworkInfo networkInfo = cm.getActiveNetworkInfo();

            if (networkInfo != null && networkInfo.isConnected()) {
                return true;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }

    @Override
    public void onStart() {
        super.onStart();
        if (!isNetworkAvailable(this)) {
            Toast.makeText(this, "Перевірте, будь ласка, підключення до мережі",
Toast.LENGTH_SHORT).show();
        }
    }

    public class BackgroundJob extends AsyncTask<Void, Void, Void> {
        @Override
        protected Void doInBackground(Void... voids) {
            try {
                Thread.sleep(4800);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            return null;
        }

        @Override
        protected void onPostExecute(Void avoid) {
            progressDialog.cancel();
        }
    }
}
```

```
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.graphics.Color;
import android.graphics.Typeface;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.Bundle;
import android.os.Handler;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v4.app.FragmentManager;
import android.support.v4.content.res.ResourcesCompat;

import android.support.v7.app.ActionBar;
import android.support.v7.app.AlertDialog;
import android.util.DisplayMetrics;
import android.util.Log;
import android.view.View;
import android.support.design.widget.NavigationView;
import android.support.v4.view.GravityCompat;
import android.support.v4.widget.DrawerLayout;
import android.support.v7.app.ActionBarDrawerToggle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.ImageView;
import android.widget.RelativeLayout;
import android.widget.TextView;
import android.widget.Toast;

import com.gadortechnologies.healthygador_android_rev3.Other.User;
import com.gadortechnologies.healthygador_android_rev3.UserPlace.CallDoctor;
import com.gadortechnologies.healthygador_android_rev3.UserPlace.ContactDoctor;
import com.gadortechnologies.healthygador_android_rev3.UserPlace.MyWrite;
import com.gadortechnologies.healthygador_android_rev3.UserPlace.Profile;
import com.gadortechnologies.healthygador_android_rev3.UserPlace.Settings;
import com.gadortechnologies.healthygador_android_rev3.UserPlace.WriteDoctor;
import com.github.johnpersano.supertoasts.library.Style;
import com.github.johnpersano.supertoasts.library.SuperActivityToast;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.google.firebase.iid.FirebaseInstanceId;

import java.util.EventListener;
import java.util.Map;
import java.util.Objects;

public class UserGrantedActivity extends AppCompatActivity
        implements NavigationView.OnNavigationItemSelectedListener {
        TextView mTitle;
        private FirebaseAuth mAuth;
        private boolean doubleBackToExitPressedOnce = false;
```

```
        private DatabaseReference mFirebaseDatabase;
        private FirebaseDatabase mFirebaseInstance;
        private String userPhoneGlob;
        private String userNameSurGlob;
        private String userSexGlob;
        //private ValueEventListener  mListener;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_user_granted);
        mAuth= FirebaseAuth.getInstance();
        //region [setTitle]
        mTitle = (TextView) findViewById(R.id.toolbar_title);
        Typeface typeface = ResourcesCompat.getFont(getApplicationContext(),
R.font.raleway_semisold);
        mTitle.setTypeface(typeface);
        //endregion

        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);


        DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
        ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
                this, drawer, toolbar, R.string.navigation_drawer_open,
R.string.navigation_drawer_close);
        drawer.addDrawerListener(toggle);
        toggle.syncState();

        NavigationView navigationView = (NavigationView) findViewById(R.id.nav_view);
        navigationView.setNavigationItemSelectedListener(this);
        navigationView.setCheckedItem(R.id.nav_profile);
        onNavigationItemSelected(navigationView.getMenu().getItem(0));

    }

    @Override
    public void onStart(){
        super.onStart();
        if (isNetworkAvailable(this))
        {

            FirebaseUser currentUser= mAuth.getCurrentUser();
            if (currentUser==null){
                Intent authIntent = new Intent(UserGrantedActivity.this,
AuthActivity.class);
                startActivity(authIntent);
                finish();
            }
            else {
                //region DATA/ANDROIDID/PHONE/FCM
                String phone =
Objects.requireNonNull(FirebaseAuth.getInstance().getCurrentUser()).getPhoneNumber();
                String phonewth = phone.replace("+", "");
                String fcm = FirebaseInstanceId.getInstance().getToken();
                //endregion
                //region До бази
```

```
                DatabaseReference FCM_phone =
FirebaseDatabase.getInstance().getReference("guestwall/" + phonewth);
                FCM_phone.child("phone").setValue(phone);
                FCM_phone.child("FCM").setValue(fcm);
                FCM_phone.child("device").setValue(deviceControl());
                //Toast.makeText(this, phone, Toast.LENGTH_SHORT).show();

                //endregion

            }
        }
        else {
            Toast.makeText(this, "Перевірте, будь ласка, підключення до мережі",
Toast.LENGTH_SHORT).show();
        }
    }

    @Override
    public void onResume(){
        super.onResume();
       String phone= FirebaseAuth.getInstance().getCurrentUser().getPhoneNumber();
        addUserChangeListener(phone);

    }

    @Override
    public void onPause(){
        super.onPause();
       // Toast.makeText(this, "Пауза", Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onDestroy(){
        super.onDestroy();
        //Toast.makeText(this, "Killed", Toast.LENGTH_SHORT).show();
    }

    private void addUserChangeListener(String userphone) {
        String phoneUser=userphone.replace("+", "");
        mFirebaseInstance = FirebaseDatabase.getInstance();
        mFirebaseDatabase = mFirebaseInstance.getReference("guestwall");
        // User data change listener
        mFirebaseDatabase.child(phoneUser).child("userdata").addValueEventListener(new
ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                User user = dataSnapshot.getValue(User.class);

                if (user == null) {
                    Log.e("Error", "User data is null!");
                    return;
                }
                //Toast.makeText(UserGrantedActivity.this, user.name + ", " +
user.surname+", "+user.number, Toast.LENGTH_SHORT).show();
                userPhoneGlob=user.number;
                userNameSurGlob=user.surname+" "+user.name;
                userSexGlob=user.sex;
                NavigationView navigationView = (NavigationView)
findViewById(R.id.nav_view);
```

```
            // navigationView.setNavigationItemSelectedListener(this);
             View headerView = navigationView.getHeaderView(0);
             TextView namesur = (TextView) headerView.findViewById(R.id.namesur);
             TextView phonenumber = (TextView) headerView.findViewById(R.id.phone);
             ImageView image = (ImageView) headerView.findViewById(R.id.photo);
             namesur.setText(userNameSurGlob);
             phonenumber.setText(userPhoneGlob);
             if (userSexGlob.equals("neutral")){
                 image.setImageResource(R.drawable.ic_navicon_neutral);
             }
             else if (userSexGlob.equals("boy")){
                 image.setImageResource(R.drawable.ic_navicon_boy);
             }
             else if (userSexGlob.equals("girl")){
                 image.setImageResource(R.drawable.ic_navicon_girl);
             }
         }

         @Override
         public void onCancelled(DatabaseError error) {
             // Failed to read value
             Log.e("Erorr", "Failed to read user", error.toException());
         }
     });
}


public String deviceControl(){
    String result;
    DisplayMetrics metrics = new DisplayMetrics();
    this.getWindowManager().getDefaultDisplay().getMetrics(metrics);

    float yInches= metrics.heightPixels/metrics.ydpi;
    float xInches= metrics.widthPixels/metrics.xdpi;
    double diagonalInches = Math.sqrt(xInches*xInches + yInches*yInches);
    if (diagonalInches>=6.5){
        result="tablet";
    }else{
        result="smartphone";
    }
    return result;
}

public static boolean isNetworkAvailable(Context con) {
    try {
        ConnectivityManager cm = (ConnectivityManager) con
                .getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo networkInfo = cm.getActiveNetworkInfo();

        if (networkInfo != null && networkInfo.isConnected()) {
            return true;
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return false;
}

@Override
```

```
    public void onBackPressed() {
        DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
        if (drawer.isDrawerOpen(GravityCompat.START)) {
            drawer.closeDrawer(GravityCompat.START);
        } else {
            //
            if (doubleBackToExitPressedOnce) {
                super.onBackPressed();
                //this.finishAffinity();
                finish();
                //System.exit(0);
                return;
            }

           //super.onBackPressed();
            this.doubleBackToExitPressedOnce = true;
            SuperActivityToast.create(UserGrantedActivity.this, new Style(),
Style.TYPE_PROGRESS_CIRCLE)
                    .setFrame(Style.FRAME_LOLLIPOP)
                    .setText(getString(R.string.toast_exit))
                    .setAnimations(Style.ANIMATIONS_POP)
                    .setDuration(Style.DURATION_SHORT)
                    .setColor(Color.rgb(0,150,136))
                    .show();
            new Handler().postDelayed(new Runnable() {
                @Override
                public void run() {
                    doubleBackToExitPressedOnce=false;
                }
            }, 2000);
        }
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.user_granted, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }

    @SuppressWarnings("StatementWithEmptyBody")
    @Override
    public boolean onNavigationItemSelected(MenuItem item) {
        // Handle navigation view item clicks here.
        int id = item.getItemId();
```

```
        if (id == R.id.nav_profile) {
            mTitle.setText("Профіль");
            Profile profile= new Profile();
            FragmentManager fm= getSupportFragmentManager();
            fm.beginTransaction().replace(R.id.fragment, profile).commit();
        } else if (id == R.id.nav_write) {
            mTitle.setText("Записатись до лікаря");
            WriteDoctor writedoctor= new WriteDoctor();
            FragmentManager fm= getSupportFragmentManager();
            fm.beginTransaction().replace(R.id.fragment, writedoctor).commit();
        } else if (id == R.id.nav_call) {
            mTitle.setText("Викликати лікаря");
            CallDoctor calldoctor= new CallDoctor();
            FragmentManager fm= getSupportFragmentManager();
            fm.beginTransaction().replace(R.id.fragment, calldoctor).commit();
        } else if (id == R.id.nav_contacts) {
            mTitle.setText("Контакти");
            ContactDoctor contactdoctor= new ContactDoctor();
            FragmentManager fm= getSupportFragmentManager();
            fm.beginTransaction().replace(R.id.fragment, contactdoctor).commit();
        } else if (id == R.id.nav_mywr) {
            mTitle.setText("Мої записи");
            MyWrite mywrite= new MyWrite();
            FragmentManager fm= getSupportFragmentManager();
            fm.beginTransaction().replace(R.id.fragment, mywrite).commit();
        } else if (id == R.id.nav_set) {
            mTitle.setText("Налаштування");
            Settings settings= new Settings();
            FragmentManager fm= getSupportFragmentManager();
            fm.beginTransaction().replace(R.id.fragment, settings).commit();
        }else if (id == R.id.exit) {
            dialog();
        }
        DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
        drawer.closeDrawer(GravityCompat.START);
        return true;
    }
    private void sendToAuth() {
        Intent authIntent= new Intent(UserGrantedActivity.this, AuthActivity.class);
        startActivity(authIntent);
        finish();
    }
    private void dialog(){
        new AlertDialog.Builder(this)
                .setTitle("Вихід")
                .setMessage("Ви дійсно бажаєте вийти з акаунту?")
                .setNegativeButton("Ні", null)
                .setPositiveButton("Так", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialogInterface, int i) {
                        mAuth.signOut();
                        sendToAuth();
                    }
                }).create().show();
    }
}
```

Index.js – файл, в якому описано процеси передачі даних на сервер.

```
'use strict';
let functions = require('firebase-functions');
let admin = require('firebase-admin');
admin.initializeApp(functions.config().firebase);

exports.NotificationCreate =
functions.database.ref('/message/{pushId}').onCreate((event) => {
      const post=event.data.val();
      const fcm= post.fcm;
      const code= post.code;
      const messageBody= "Ваш код: " + code;
      const messageTitle="Аутентифікація";

      console.log("Токен: ", fcm);
      console.log("Код: ", code);
      console.log("Загаловок: ", messageTitle);
      console.log("Тіло: ", messageBody);
      if (!event.data.val()){
            return console.log('fail!');
      }

      const payload= {
            notification:{
                  title : messageTitle,
                  body: messageBody,
                  icon: "default"
            }
      };

      return admin.messaging().sendToDevice(fcm, payload).then(response =>{

            return console.log('Повідомлення надіслано');
      });
      });

 exports.NotificationUpdate =
functions.database.ref('/message/{pushId}').onUpdate((event) => {
      const post=event.data.val();
      const fcm= post.fcm;
      const code= post.code;
      const messageBody= "Ваш код: " + code;
      const messageTitle="Аутентифікація";

      console.log("Токен: ", fcm);
      console.log("Код: ", code);
      console.log("Загаловок: ", messageTitle);
      console.log("Тіло: ", messageBody);
      if (!event.data.val()){
            return console.log('truble');
      }

      const payload= {
            notification:{
                  title : messageTitle,
                  body: messageBody,
                  icon: "default"
            }
      };
```

```
      return admin.messaging().sendToDevice(fcm, payload).then(response =>{

            return console.log('Повідомлення надіслано');
      });
      });

      exports.statusUpdate =
functions.database.ref('/guestwall/{userId}').onUpdate((event) => {
            const post= event.data.val();
            const pcstat=post.PCstatus;

            console.log(pcstat);

            if (pcstat!=='authorized')
            {
                  return console.log('Ніяких змін');
            }

            const fcm=post.FCM;

            const payload= {
            notification:{
                  title : 'Вітаю',
                  body: 'Ви авторизувалися у Windows-клієнті!',
                  icon: "default"
            }
      };

            return admin.messaging().sendToDevice(fcm, payload).then(response =>{

            return console.log('Повідомлення надіслано');
      });
      });

            exports.statusDisUpdate =
functions.database.ref('/guestwall/{userId}').onUpdate((event) => {
            const post= event.data.val();
            const pcstat=post.PCstatus;

            console.log(pcstat);

            if (pcstat!=='disauth')
            {
                  return console.log('Ніяких змін');
            }

            const fcm=post.FCM;

            const payload= {
            notification:{
                  title : 'Увага',
                  body: 'Ви вийшли з акаунту Windows-клієнта!',
                  icon: "default"
            }
      };

            return admin.messaging().sendToDevice(fcm, payload).then(response =>{

            return console.log('Повідомлення надіслано');
```

```
      });
      });

      exports.userData = functions.database.ref('/guestwall/{userId}').onCreate((event)
=> {
            const teleph=event.params.userId;
            const a="(";
            const b=")";
            const c=" ";
            const d="+"
            const mydata=d+ teleph.substr(0, 3) +c+ a +
teleph.substr(2,3)+b+c+teleph.substr(5,3)+c+teleph.substr(8,2)+c+teleph.substr(10,2);
            const userref=admin.database().ref('guestwall/'+teleph)
      return userref.update({
      userdata: {
    name: "Ім`я",
      surname:"Прізвище",
    number: mydata,
      sex:"neutral",
      birth:"",
      home:""
      }
            });
      });
```

```
package com.gadortechnologies.healthygador_android_rev3.UserPlace;


import android.graphics.Color;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;

import com.gadortechnologies.healthygador_android_rev3.R;
import com.github.johnpersano.supertoasts.library.Style;
import com.github.johnpersano.supertoasts.library.SuperActivityToast;

/**
 * A simple {@link Fragment} subclass.
 */
public class WriteDoctor extends Fragment {

    public WriteDoctor() {

    }


    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
        View m = inflater.inflate(R.layout.fragment_write_doctor, container, false);

        Button btn= (Button) m.findViewById(R.id.searchBtn);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View view) {
                SuperActivityToast.create(getContext(), new Style(),
Style.TYPE_STANDARD)
                        .setFrame(Style.FRAME_LOLLIPOP)
                        .setText("Модуль у розробці")
                        .setAnimations(Style.ANIMATIONS_POP)
                        .setDuration(Style.DURATION_MEDIUM)
                        .setColor(Color.rgb(0,150,136))
                        .show();
            }
        });


        return m;
    }

}
```

```
package com.gadortechnologies.healthygador_android_rev3.UserPlace;


import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import com.gadortechnologies.healthygador_android_rev3.R;

/**
 * A simple {@link Fragment} subclass.
 */
public class Settings extends Fragment {


    public Settings() {
        // Required empty public constructor
    }


    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        View h = inflater.inflate(R.layout.fragment_settings, container, false);

        return  h;
    }

}
```

```
package com.gadortechnologies.healthygador_android_rev3.UserPlace;


import android.graphics.Color;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.design.widget.NavigationView;
```

```
import android.support.v4.app.Fragment;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;

import com.gadortechnologies.healthygador_android_rev3.Other.User;
import com.gadortechnologies.healthygador_android_rev3.R;
import com.github.johnpersano.supertoasts.library.Style;
import com.github.johnpersano.supertoasts.library.SuperActivityToast;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.google.firebase.iid.FirebaseInstanceId;

import java.util.Objects;


public class Profile extends Fragment {
    EditText name;
    EditText phone;
    EditText surname;
    EditText birth;
    EditText address;
    Spinner sex;

    private DatabaseReference mFirebaseDatabase;
    private FirebaseDatabase mFirebaseInstance;
    private String userPhoneGlob;
    private String userNameGlob;
    private String userSurnGlob;
    private String userBirthGlob;
    private String userAddressGlob;
    private String userSexGlob;
    private String selecteditem;
    public Profile() {
        // Required empty public constructor
    }


    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {

        View v = inflater.inflate(R.layout.fragment_profile, container, false);
        name= (EditText) v.findViewById(R.id.name);
        surname =(EditText) v.findViewById(R.id.surname);
        phone= (EditText) v.findViewById(R.id.phone);
        birth = (EditText) v.findViewById(R.id.birth);
```

```
        address= (EditText) v.findViewById(R.id.address);
        sex = (Spinner) v.findViewById(R.id.sex);

         final String colors[] = {"Гендер","Хлопець","Дівчина"};
        ArrayAdapter<String> myAdaptor= new
ArrayAdapter<String>(Objects.requireNonNull(getContext()),
android.R.layout.simple_list_item_1, colors);

myAdaptor.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        sex.setAdapter(myAdaptor);
        sex.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView<?> adapterView, View view, int i,
long l) {
                switch (i)
                {
                    case 0:
                        userSexGlob = "neutral";

                        break;
                    case 1:
                        userSexGlob = "boy";
                        break;
                    case 2:
                        userSexGlob = "girl";
                        break;
                }
            }

            @Override
            public void onNothingSelected(AdapterView<?> adapterView) {

            }
        });
        // String phone= FirebaseAuth.getInstance().getCurrentUser().getPhoneNumber();
        // addUserChangeListener(phone);

        Button btn= (Button) v.findViewById(R.id.save);
        btn.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View arg0) {
                String phone =
FirebaseAuth.getInstance().getCurrentUser().getPhoneNumber();
                String phonewth = phone.replace("+", "");
                String fcm = FirebaseInstanceId.getInstance().getToken();
                String userNameStr=name.getText().toString();
                String userSurnStr=surname.getText().toString();
                String userHomeStr=address.getText().toString();
                String userBirthStr=birth.getText().toString();
                DatabaseReference FCM_phone =
FirebaseDatabase.getInstance().getReference("guestwall/" + phonewth+"/userdata");
                FCM_phone.child("name").setValue(userNameStr);
                FCM_phone.child("surname").setValue(userSurnStr);
                FCM_phone.child("home").setValue(userHomeStr);
                FCM_phone.child("birth").setValue(userBirthStr);
                FCM_phone.child("sex").setValue(userSexGlob);
                //
```

```
                SuperActivityToast.create(getContext(), new Style(),
Style.TYPE_PROGRESS_BAR)
                        .setFrame(Style.FRAME_LOLLIPOP)
                        .setText(getString(R.string.toast_well))
                        .setAnimations(Style.ANIMATIONS_POP)
                        .setDuration(Style.DURATION_MEDIUM)
                        .setColor(Color.rgb(0,150,136))
                        .show();
            }
        });
        return v;

    }
    @Override
    public void onStart(){
         super.onStart();


    }

    @Override
    public void onResume(){
        super.onResume();
        String phone= FirebaseAuth.getInstance().getCurrentUser().getPhoneNumber();
        addUserChangeListener(phone);
    }

    private void addUserChangeListener(String userphone) {
        String phoneUser=userphone.replace("+", "");
        mFirebaseInstance = FirebaseDatabase.getInstance();
        mFirebaseDatabase = mFirebaseInstance.getReference("guestwall");
        // User data change listener
        mFirebaseDatabase.child(phoneUser).child("userdata").addValueEventListener(new
ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                User user = dataSnapshot.getValue(User.class);

                if (user == null) {
                    Log.e("Error", "User data is null!");
                    return;
                }
                //Toast.makeText(UserGrantedActivity.this, user.name + ", " +
user.surname+", "+user.number, Toast.LENGTH_SHORT).show();
                userPhoneGlob=user.number;
                userNameGlob=user.name;
                userSurnGlob=user.surname;
                userAddressGlob=user.home;
                userBirthGlob=user.birth;
                userSexGlob=user.sex;
                String userBirthGlob_modificator=userBirthGlob.replace(".", "");

                name.setText(userNameGlob);
                surname.setText(userSurnGlob);
                phone.setText(userPhoneGlob);
                birth.setText(userBirthGlob_modificator);
                address.setText(userAddressGlob);
                phone.setEnabled(false);
                if (userSexGlob.equals("neutral")){
```

```
                    sex.setSelection(0);
                }
                else if (userSexGlob.equals("boy")){
                    sex.setSelection(1);
                }
                else if (userSexGlob.equals("girl")){
                   sex.setSelection(2);
                }
            }

            @Override
            public void onCancelled(DatabaseError error) {
                // Failed to read value
                Log.e("Erorr", "Failed to read user", error.toException());
            }
        });
    }

}
```

```
package com.gadortechnologies.healthygador_android_rev3.UserPlace;


import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import com.gadortechnologies.healthygador_android_rev3.R;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

import java.util.Map;

/**
 * A simple {@link Fragment} subclass.
 */
public class MyWrite extends Fragment {

    TextView textView;
    TextView textView2;
    TextView textView3;
    TextView textView4;
    TextView textView5;
    TextView textView6;
    public MyWrite() {
        // Required empty public constructor
    }


    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
```

```
        // Inflate the layout for this fragment
        View g = inflater.inflate(R.layout.fragment_my_write, container, false);
        textView = (TextView) g.findViewById(R.id.textView);
        textView2 = (TextView) g.findViewById(R.id.textView2);

        textView3 = (TextView) g.findViewById(R.id.textView3);
        textView4 = (TextView) g.findViewById(R.id.textView4);

        textView5 = (TextView) g.findViewById(R.id.textView5);
        textView6 = (TextView) g.findViewById(R.id.textView6);
        String phone= FirebaseAuth.getInstance().getCurrentUser().getPhoneNumber();
        getData(phone, "0", textView, textView2);
        getData(phone, "1", textView3, textView4);
        getData(phone, "2", textView5, textView6);
        return g;
    }

    public void getData(String userphone, String node, final TextView tx1, final
TextView tx2) {
        String phoneUser=userphone.replace("+", "");
        FirebaseDatabase.getInstance().getReference()
                .child("guestwall")
                .child(phoneUser)
                .child("mywrite")
                .child(node)
                .addValueEventListener(new ValueEventListener() {
                    @Override
                    public void onDataChange(DataSnapshot dataSnapshot) {
                        @SuppressWarnings("unchecked")
                        Map<String, Object> map = (Map<String, Object>)
dataSnapshot.getValue();
                        String address = (String) map.get("address");
                        String date = (String) map.get("date");
                        String docmidname = (String) map.get("docmidname");
                        String docname = (String) map.get("docname");
                        String docsurname = (String) map.get("docsurname");
                        String spec = (String) map.get("spec");
                        String time = (String) map.get("time");

                        tx1.setText(docsurname +" "+docname +" " +docmidname + "\n ("
+spec+")");
                        tx2.setText("Адреса: " + address+ "\n"
                        +"Прийом: "+ date + " " +time);
                    }
                    @Override
                    public void onCancelled(DatabaseError databaseError) {/*Do
Nothing*/}
                });

    }

}
```

```
package com.gadortechnologies.healthygador_android_rev3.UserPlace;


import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
```

```
import android.support.annotation.NonNull;
import android.support.v4.app.Fragment;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.TextView;
import android.widget.Toast;

import com.gadortechnologies.healthygador_android_rev3.R;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

/**
 * A simple {@link Fragment} subclass.
 */
public class ContactDoctor extends Fragment {
    String items[]=new String[] {"", "", ""};

    public ContactDoctor() {
        // Required empty public constructor
    }


    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        View f = inflater.inflate(R.layout.fragment_contact_doctor, container, false);
        final ListView resultsListView  = (ListView) f.findViewById(R.id.list);

        HashMap<String, String> nameAddresses = new HashMap<>();
        nameAddresses.put("Швидка допомога", "+38 (067) 879 73 01");
        nameAddresses.put("Педіатр", "+38 (073) 456 55 01");
        nameAddresses.put("Терапевт", "+38 (073) 456 55 01");
       // nameAddresses.put("Donna", "789 Escort St");
       // nameAddresses.put("Bartholomew", "332 Dunkin St");
       // nameAddresses.put("Eden", "421 Angelic Blvd");

        List<HashMap<String, String>> listItems = new ArrayList<>();
        SimpleAdapter adapter = new SimpleAdapter(getContext(), listItems,
R.layout.list_item,
                new String[]{"First Line", "Second Line"},
                new int[]{R.id.text1, R.id.text2});


        Iterator it = nameAddresses.entrySet().iterator();
        while (it.hasNext())
        {
            HashMap<String, String> resultsMap = new HashMap<>();
            Map.Entry pair = (Map.Entry)it.next();
```

```
            resultsMap.put("First Line", pair.getKey().toString());
            resultsMap.put("Second Line", pair.getValue().toString());
            listItems.add(resultsMap);
        }

        resultsListView.setAdapter(adapter);
        resultsListView .setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> adapterView, View view, int i, long
l) {
                TextView textView = (TextView) view.findViewById(R.id.text1);
                TextView textView2 = (TextView) view.findViewById(R.id.text2);
                String data=textView.getText().toString();
                String phone=textView2.getText().toString();
                String phonecall=phone.replaceAll(" ", "");
                String phonefin=phonecall.replace("(", "");
                String phonefin2=phonefin.replace("(", "");

                Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:" +
phonefin2));
                startActivity(intent);

                Toast.makeText(getContext(), data+"\n"+phone,
Toast.LENGTH_SHORT).show();
            }
        });
        return f;
    }

}

package com.gadortechnologies.healthygador_android_rev3.UserPlace;


import android.graphics.Color;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.Spinner;

import com.gadortechnologies.healthygador_android_rev3.R;
import com.github.johnpersano.supertoasts.library.Style;
import com.github.johnpersano.supertoasts.library.SuperActivityToast;

import java.util.Objects;

/**
 * A simple {@link Fragment} subclass.
 */
public class CallDoctor extends Fragment {
 Spinner stan;
 EditText place;
```

```
 CheckBox info;
 Button callDoc;

    public CallDoctor() {
        // Required empty public constructor
    }


    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        View z = inflater.inflate(R.layout.fragment_call_doctor, container, false);
        stan=(Spinner) z.findViewById(R.id.stan);
        place=(EditText) z.findViewById(R.id.place);
        info=(CheckBox) z.findViewById(R.id.info);
        final String colors[] = {"Оцініть стан","Істотня проблема","Критичний стан"};
        ArrayAdapter<String> myAdaptor= new
ArrayAdapter<String>(Objects.requireNonNull(getContext()),
android.R.layout.simple_list_item_1, colors);

myAdaptor.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        stan.setAdapter(myAdaptor);
        callDoc= (Button) z.findViewById(R.id.callDoc);
        callDoc.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                SuperActivityToast.create(getContext(), new Style(),
Style.TYPE_STANDARD)
                        .setFrame(Style.FRAME_LOLLIPOP)
                        .setText("Модуль у розробці")
                        .setAnimations(Style.ANIMATIONS_POP)
                        .setDuration(Style.DURATION_MEDIUM)
                        .setColor(Color.rgb(0,150,136))
                        .show();
            }
        });
        return z;
    }

}
```

Додаток В

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

# ОПИС ПРОГРАМНОГО МОДУЛЮ

Листів 2

Київ – 2020

# АНОТАЦІЯ

Даний програмний продукт є мобільним медичним сервісом для видаленої взаємодії користувача із державними медичними закладами з використанням хмарних технологій. Цей додаток вимагає мобільного пристрою з операційною системою Android та стабільний доступ до Інтернету.

Цей мобільний додаток побудовано з застосуванням концепції *Material Design*. Дана концепція дозволяє розроблювати інтерфейс користувача зручним та швидким способом з використанням уніфікованих та унікальних компонентів. Це надає інтерфейсу більшої зрозумілості та простоти для будь-якого користувача.

Обмін даними реалізовано за допомогою хмарних технологій. Хмарні технології – це збірка концепцій, які передбачають надавання обчислювальних ресурсів, наприклад, для зберігання даних та їх обробки, без безпосереднього активного управління користувачем. Термін зазвичай використовується для опису центрів обробки даних, доступних багатьом користувачам через Інтернет.

Забезпечення послуг хмарних технологій відбувається за наявності мереж високої пропускної здатності, недорогих комп'ютерів та пристроїв зберігання даних . Важливим для хмарних технологій є також широке впровадження апаратної віртуалізації.

Хмарні технології дозволяють використовувати даний мобільний додаток не тільки у випадку нормального Інтернет-з’єднання, а і в умовах, коли такого з’єднання немає або воно недостатньо стабільне.

Також використання хмарних технологій оптимізує роботу загального комплексу, оскільки не вимагає виконувати сценарії на локальному пристрої, який найчастіше має недостатньо пам’яті або обчислювальної потужності.

Даний програмний продукт має цільове спрямування на сферу надання медичних послуг. Для отримання максимального результату практики використання рекомендовано інтегрувати платформу у лікувальні та профілактичні заклади.

1. Загальні відомості

Додаток *HealthYou* призначений для взаємодії пацієнтів з клініками та іншими медичними закладами. Даний програмний продукт розроблено за допомогою *Android Java* та продуктів хмарної платформи *Google*.

Зберігання даних відбувається у хмарному сховищі, а взаємодія пацієнтів з сервером відбувається за допомогою клієнтського додатку для *Android*.

2. Функціональне призначення

Даний програмний продукт дозволяє з'єднатись з будь-яким медичним закладом. Функціонал дозволяє записатись на прийом, зберегти контакти лікарів, викликати лікаря та вести свою картку пацієнта.

3. Опис логічної структури

Загальна логічна структура складається з двох компонентів: мобільного додатку та серверного додатку.

Мобільний додаток – це програмне забезпечення, яке встановлює зв'язок з хмарним сховищем і відображає запитувані користувачем дані. За допомогою цього додатку відбувається обмін даними з серверною частиною, оскільки він являє собою інтерфейс користувача для введення та виведення даних.

Серверний додаток є набором функціональних складових хмарних платформ *Google*. За допомогою функцій *Firebase* відбувається збереження, зчитування та оновлення даних, що зберігаються в базі даних.

4. Використовувані технічні засоби

Даний проект розроблено та впроваджено за допомогою інструментів середовища для розробки програмного забезпечення *Android studio* та мови *Android Java*. Для моделювання та розробки логічних діаграм станів та скінченних автоматів було використано інструмент Draw.io.

5. Виклик і завантаження

Програмний продукт *HealthYou* можливо встановити, отримавши файл встановлення APK. Цей файл можливо завантажити з офіційного магазину *Google Play Store*, коли він з’явиться там, а також завантажити з мережі Інтернет.

Серверний додаток встановлюється та налаштовується системним адміністратором або розробником, участь користувача не потрібна.

6. Вхідні дані

Первинними вхідними даними є масив рядків, який передає номер телефону, повне ім’я користувача, його вік, його медичну картку.

Вторинними вхідними даними та даними, що передаються в процесі користування, є інформація про користувачів (лікарів або адміністраторів), інформація про візити до лікаря, запис до лікаря, виклики лікаря тощо.

7. Вихідні дані

Вихідними даними є такі дані, які передаються інструментами хмарних платформ *Google*.

Так, клієнтський додаток має доступ безпосередньо до гілки даних користувача, а також до інформації, яка пов’язана з його профілем. З боку медичних закладів вихідними даними є дані про пацієнтів, які записались на прийом або викликали лікаря, а також статистичні дані про діяльність пацієнтів та лікарів.

# Додаток Г

## Опитування

| | | |
|---|---|---|
| 1 | Ваша стать? | Чоловіча |
| | | Жіноча |
| 2 | Ваш вік? | менше 18 |
| | | 18-24 |
| | | 25-35 |
| | | 35-45 |
| | | 45-55 |
| | | 55+ |
| 3 | Ваш рід діяльності | Не працюю і не вчусь |
| | | Вчусь |
| | | Працюю |
| 4 | Наскільки часто ви відвідуєте поліклініку? | Ніколи |
| | | Інколи/Якщо захворів |
| | | Регулярно/За рекомендаціями |
| 5 | Як часто ви маєте проблеми комунікації з поліклінікою? | Ніколи |
| | | Рідко |
| | | Іноді |
| | | Часто |
| | | Завжди |
| 6 | Чи стикались Ви з проблемою неможливості записатись до лікаря? | Ні |
| | | Так |
| 7 | Яка концепція мобільного додатку зручніша для Вас? | Компактний зовнішній вигляд, необхідний мінімум функціоналу |
| | | Зовнішній вигляд не важливий, потрібен широкий функціонал |

Додаток Д

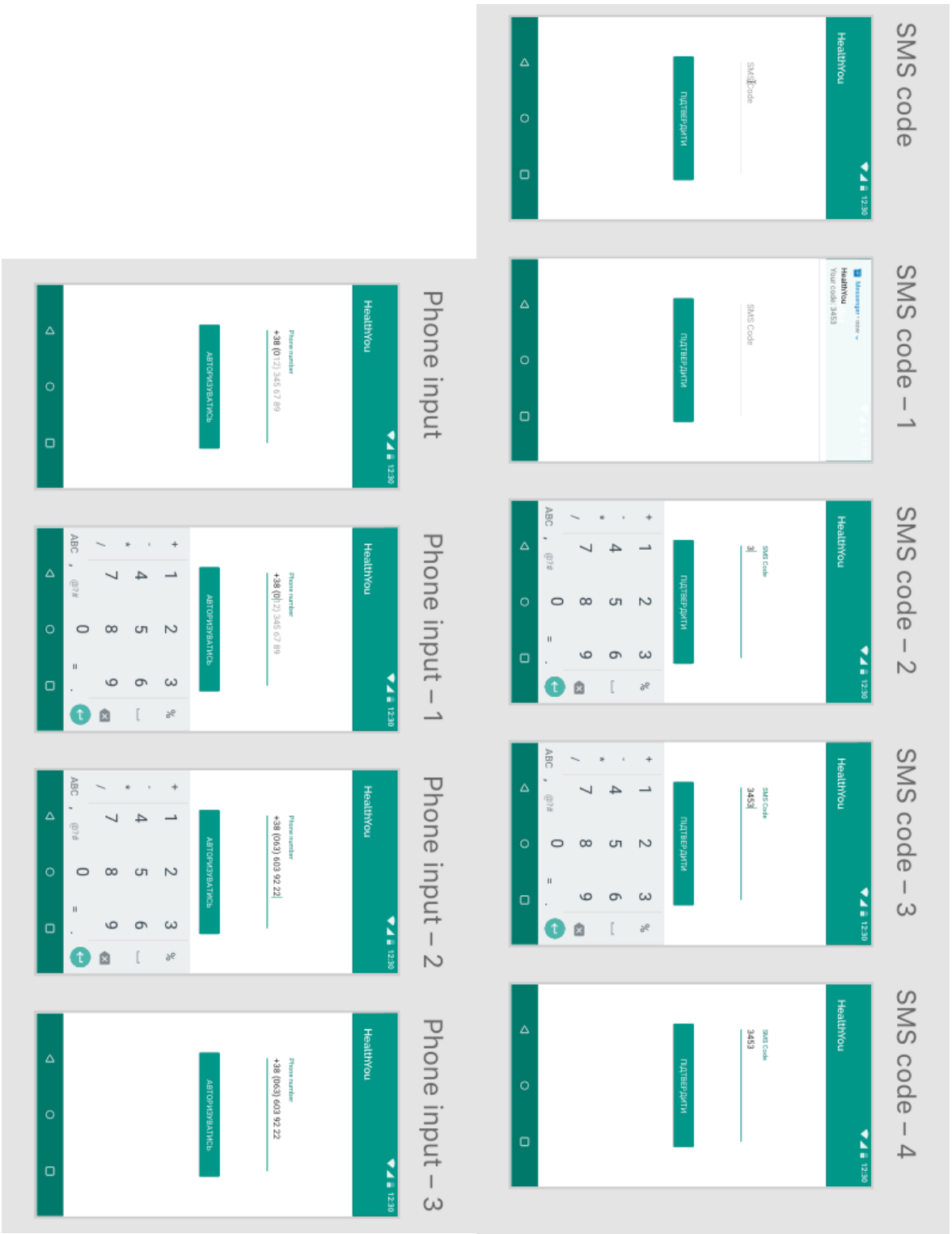



Рисунок Д.1. Авторизація

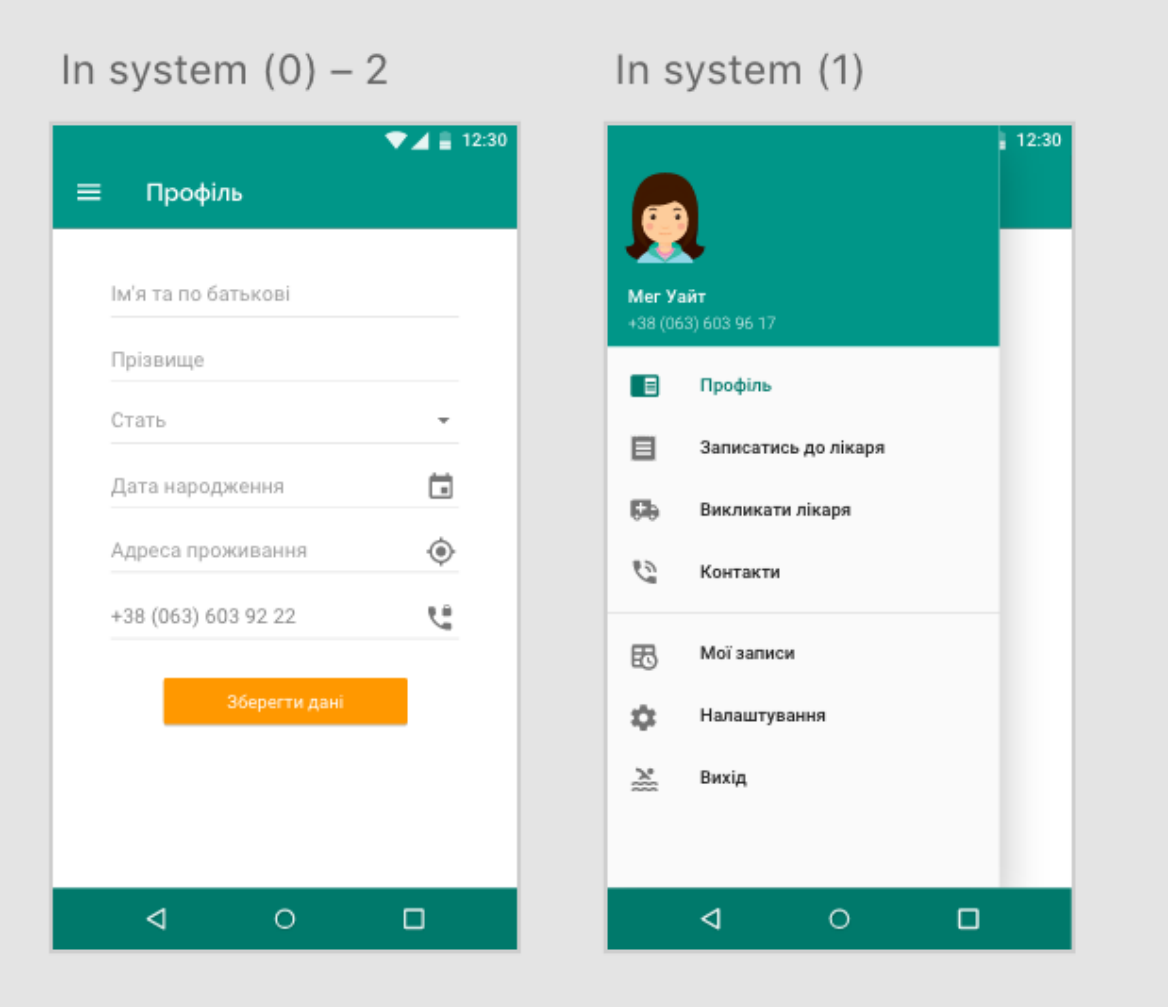



Рисунок Д.2. Профіль користувача

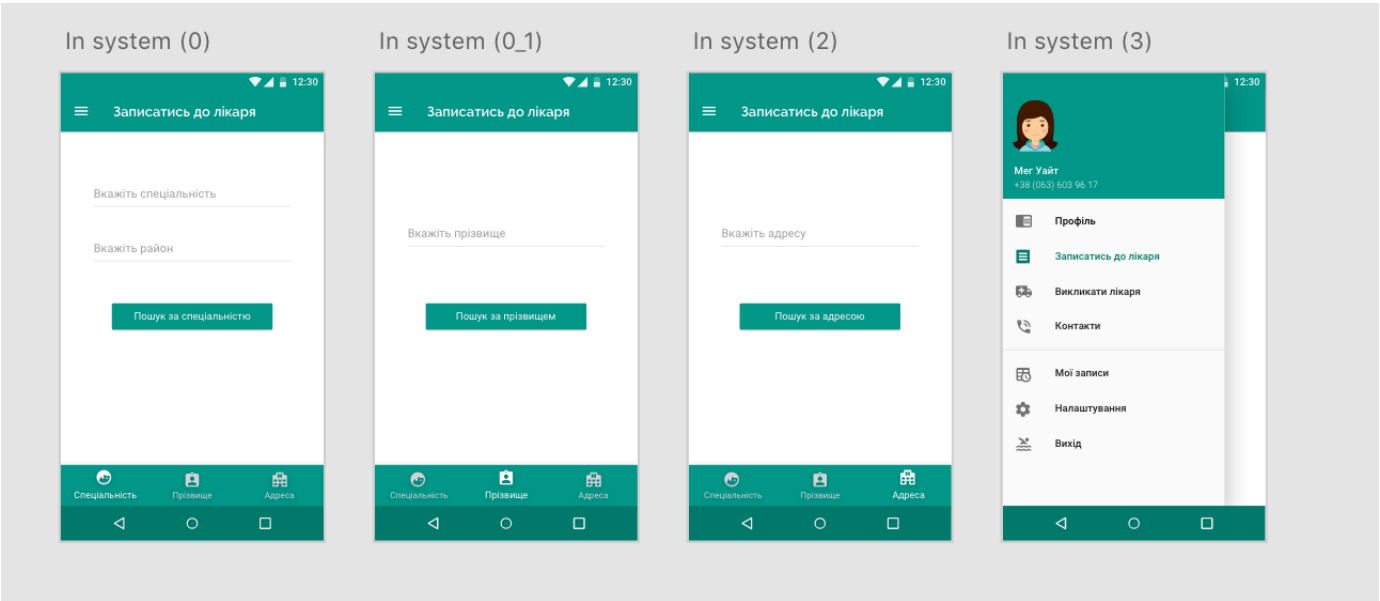



Рисунок Д.3. Запис до лікаря

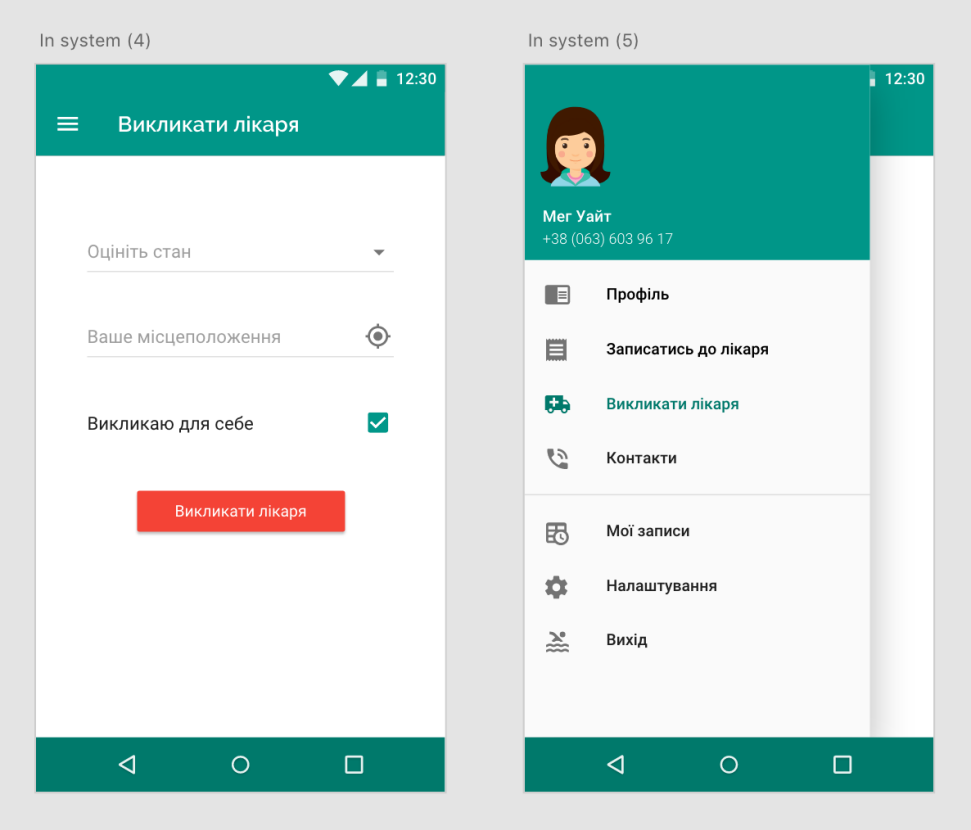



Рисунок Д.4. Виклик лікаря

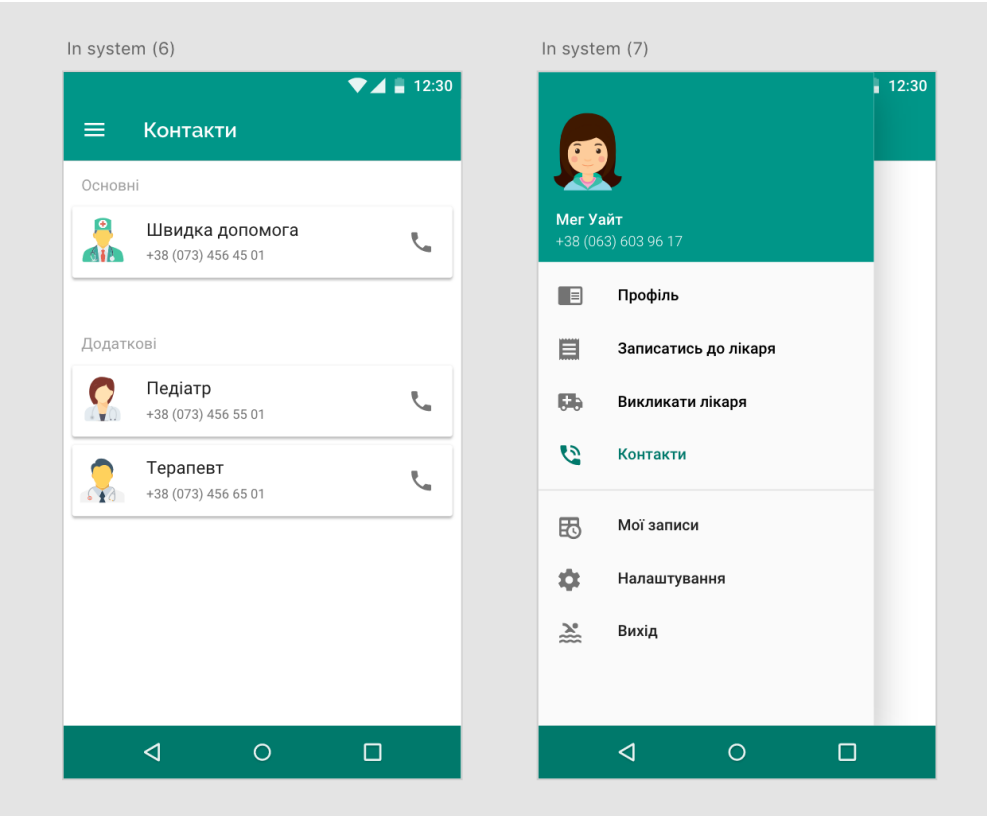



Рисунок Д.5. Контакти лікарів

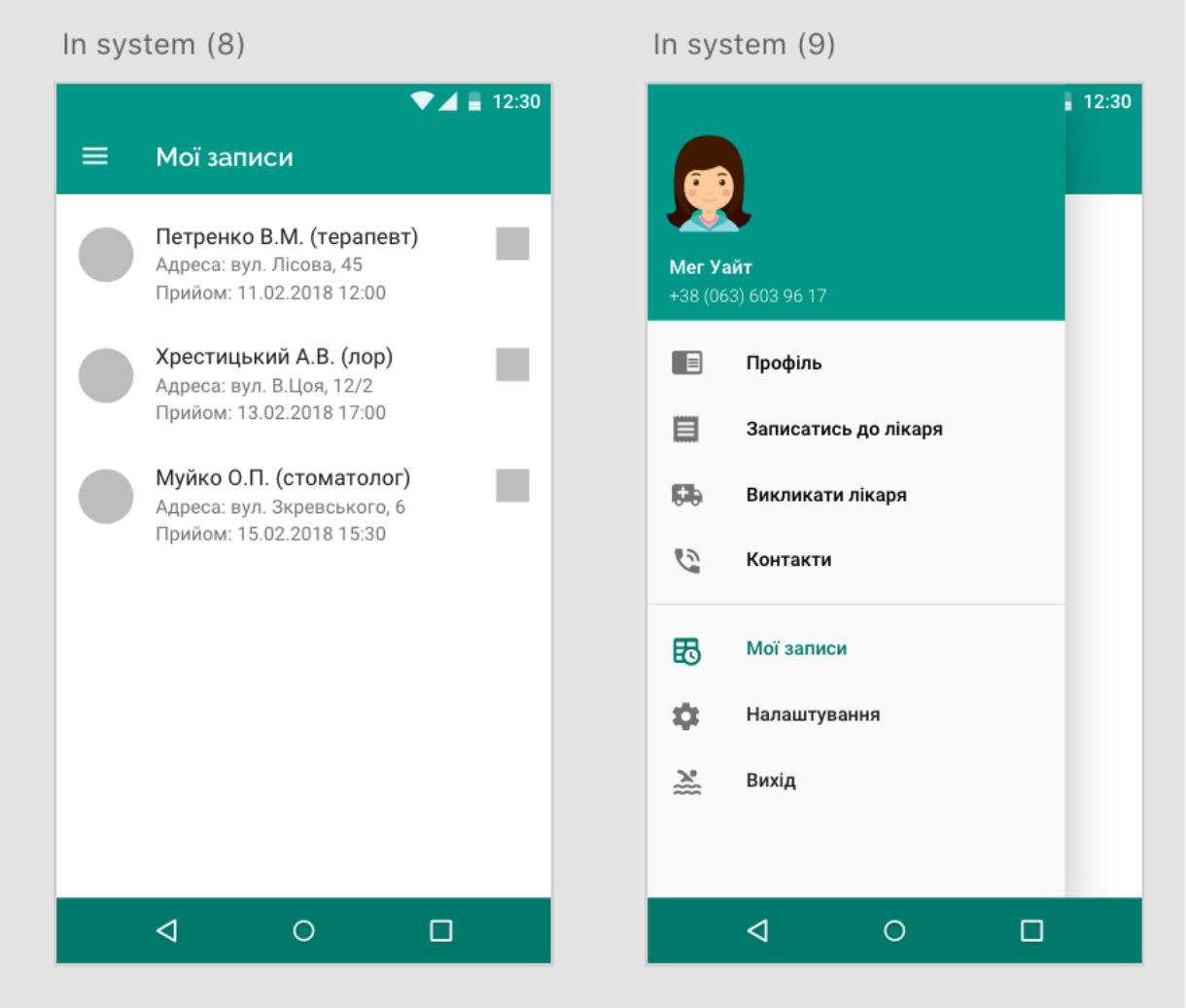



Рисунок Д.6. Записи пацієнта

Додаток 6

Схема зв’язків між екранами мобільного додатку

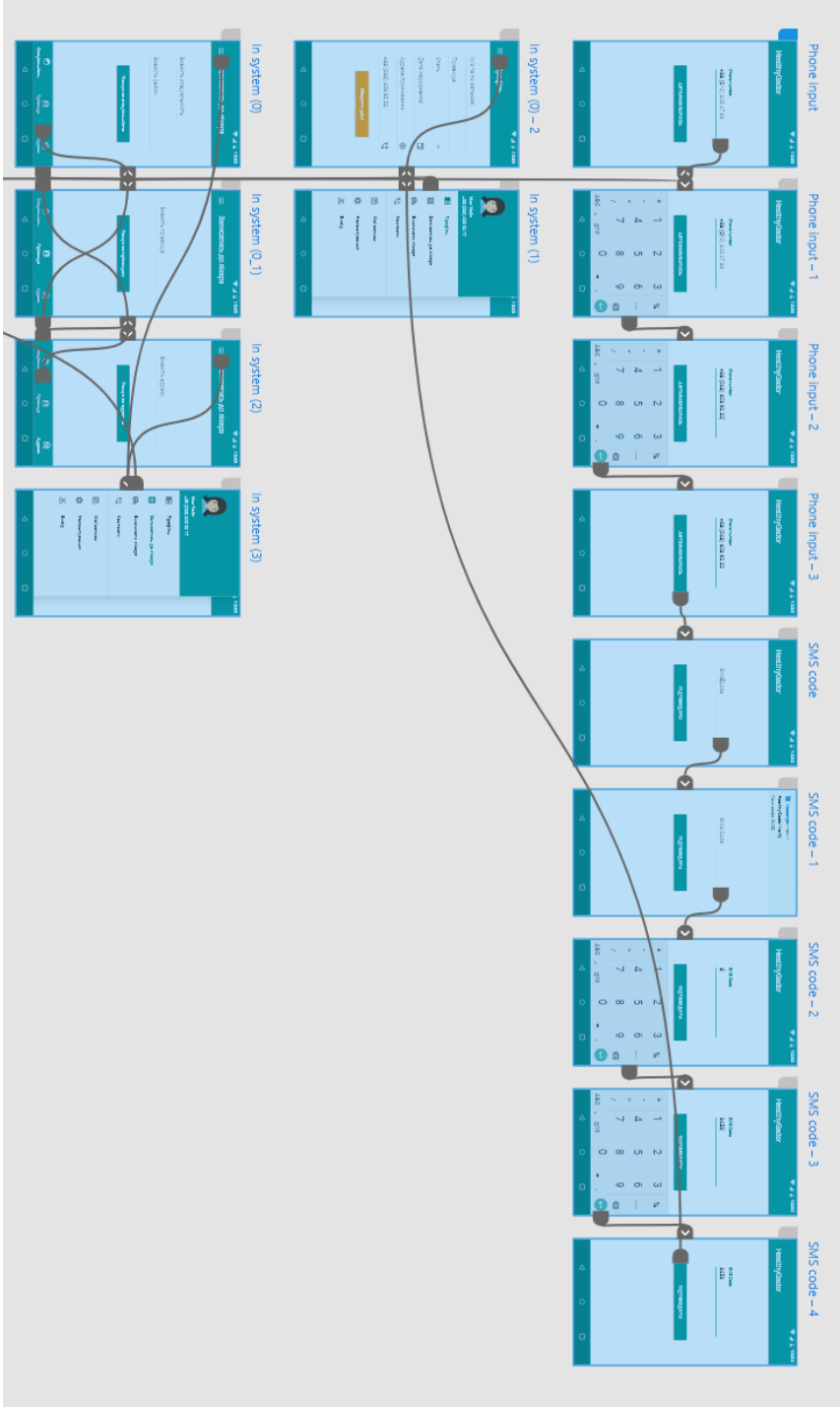



Рисунок Г.1. Взаємозв’язки між екранами

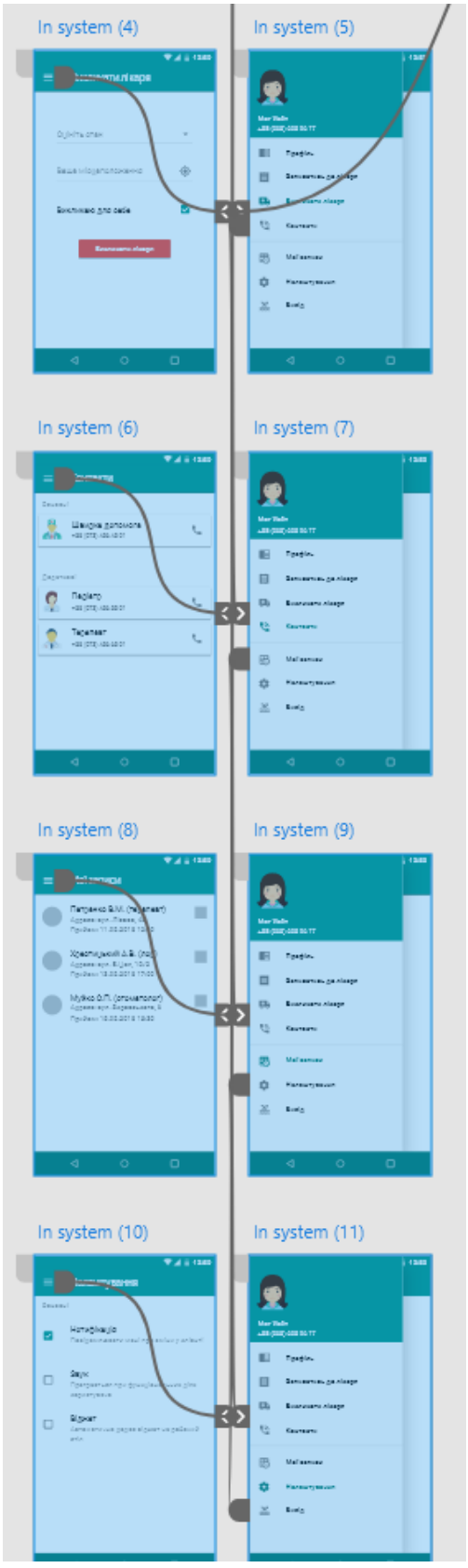



Рисунок Г.2. Взаємозв’язки між екранами